NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

PERFORMANCE ANALYSIS OF THE ADVANCED AMPHIBIOUS ASSAULT VEHICLE PERSONNEL VARIANT (AAAV-P) VETRONICS COMMUNICATIONS SYSTEM HIGH SPEED DATA BUS

by

Deborah G. Peyton

June 1999

Thesis Advisor: Raymond F. Bernstein, Jr.
Second Reader: Robert E. Parker, Jr.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
June 1999

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
PERFORMANCE ANALYSIS OF THE ADVANCED AMPHIBIOUS ASSAULT VEHICLE PERSONNEL VARIANT (AAAV-P) VETRONICS COMMUNICATIONS SYSTEM HIGH SPEED DATA BUS

5. FUNDING NUMBERS

6. AUTHOR(S)
Deborah G. Peyton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DRPM AAA
AAAV Technology Center
991 Annapolis Way
Woodbridge, VA 22191

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The United States Marine Corps Advanced Amphibious Assault Vehicle Personnel Variant (AAAV-P) is a critical weapon system that supports the Naval "Operational Maneuver from the Sea" (OMFTS) concept. The AAAV-P will provide the Marine Corps with the ability to project naval power ashore in support of strategic objectives. The Marine Corps is relying on the AAAV-P to exploit the sea and land terrain in order to attain surprise and be able to rapidly take advantage of weak points in enemy littoral defenses. The first prototype of the AAAV-P will be completed in June 1999. Successful operation of the AAAV-P is heavily dependent upon the Vetronics System communications network residing within the vehicle. The Vetronics System supports three networks: a High Speed Data Bus, a Utility Bus, and a Powertrain Bus. This thesis develops a model of the High Speed Data Bus, which is considered the main data bus within the AAAV-P. The simulation results of the model are analyzed to determine if the High Speed Data Bus is properly designed to handle the anticipated communications traffic that will traverse the network. The network performance capability is evaluated under various scenarios.

14. SUBJECT TERMS
Electronics, Data and Computer Communications, Network Modeling and Simulation, OPNET

15. NUMBER OF PAGES
205

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18, 298-102

Approved for public release; distribution is unlimited.

PERFORMANCE ANALYSIS OF THE ADVANCED AMPHIBIOUS ASSAULT VEHICLE PERSONNEL VARIANT (AAAV-P) VETRONICS COMMUNICATIONS SYSTEM HIGH SPEED DATA BUS

Deborah G. Peyton
B.S., University of Rochester, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1999

Author: Deborah G. Peyton
Deborah G. Peyton

Approved by: Raymond F. Bernstein Jr
Raymond F. Bernstein, Jr., Thesis Advisor

Robert E. Parker Jr
Robert E. Parker, Jr., Second Reader

Jeffrey B. Knorr
Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

The United States Marine Corps Advanced Amphibious Assault Vehicle Personnel Variant (AAAV-P) is a critical weapon system that supports the Naval "Operational Maneuver from the Sea" (OMFTS) concept. The AAAV-P will provide the Marine Corps with the ability to project naval power ashore in support of strategic objectives. The Marine Corps is relying on the AAAV-P to exploit the sea and land terrain in order to attain surprise and be able to rapidly take advantage of weak points in enemy littoral defenses. The first prototype of the AAAV-P will be completed in June 1999. Successful operation of the AAAV-P is heavily dependent upon the Vetronics System communications network residing within the vehicle. The Vetronics System supports three networks: a High Speed Data Bus, a Utility Bus, and a Powertrain Bus. This thesis develops a model of the High Speed Data Bus, which is considered the main data bus within the AAAV-P. The simulation results of the model are analyzed to determine if the High Speed Data Bus is properly designed to handle the anticipated communications traffic that will traverse the network. The network performance capability is evaluated under various scenarios.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

The United States Marine Corps Advanced Amphibious Assault Vehicle (AAAV) is a critical weapon system that supports the Naval "Operational Maneuver from the Sea" (OMFTS) concept. The AAAV will provide the Marine Corps with the ability to project naval power ashore in support of strategic objectives. According to General C. C. Krulak, USMC, Commandant of the Marine Corps, it "will virtually revolutionize every facet of the Marine Corps Combat Operations." The Marine Corps is relying on the AAAV to exploit the sea and land terrain in order to attain surprise and be able to rapidly take advantage of weak points in enemy littoral defenses. [1]

Successful operation of the AAAV is heavily dependent upon the Vetronics System communications network residing within the vehicle. The Vetronics System, within the AAAV Personnel Variant (AAAV-P), supports three networks: a High Speed Data Bus, a Utility Bus, and a Powertrain Bus. This thesis will focus on the modeling and simulation of the High Speed Data Bus network to ensure that it is properly designed to handle the anticipated communications traffic that will traverse the network.

A. OBJECTIVES

The objective of this thesis is to develop an OPNET model to simulate the High Speed Data Bus network within the AAAV-P [2]. The modeling and simulation tool utilized is MIL3's Optimized Network Engineering Tool (OPNET), version 5.1D. The model is simulated to determine if there are any performance issues associated with the network. The simulation results will help validate the performance capability of the Data

Bus under various scenarios. Further, changes to the network and additional modeling efforts are recommended to improve efficiency.

B. THESIS ORGANIZATION

Chapter II provides background on the AAAV-P and its Vetronics System communication network. The High Speed Data Bus is discussed in detail. In addition, the reader is provided with a general understanding of the Utility and Powertrain Buses.

Chapter III provides a detailed discussion on the network model developed and the associated code written for this thesis.

Chapter IV identifies the simulation scenarios that were used to analyze the performance capability of the Vetronics System High Speed Data Bus network model.

Chapter V describes the results obtained through modeling and simulation of the network and provides an analysis of the results.

Chapter VI presents the conclusions and recommendations.

Several appendices are included to provide specific technical data necessary to fully understand the modeling efforts. Appendix A contains a list of acronyms and abbreviations used throughout this thesis. Appendix B contains the High Speed Data Bus frame formats. Appendix C contains an overview of OPNET, the software tool used in the modeling and simulation efforts. Appendix D contains the message generator code produced in the course of this research. Similarly, Appendix E contains the message receiver code produced in the course of this research. Appendix F contains an excerpt of the source document provided by the AAAV program office for the message traffic used to test the capability of the High Speed Data Bus. Appendix G contains the general data files (GDFs) identifying the messages that comprise the traffic load, and associated parameters, placed on the High

Speed Data Bus during the simulations. Appendix H contains the modified code used to collect user-defined statistics during the simulation process.

II. BACKGROUND

The Advanced Amphibious Assault Vehicle (AAAV) is a self-deploying, nuclear, biological, and chemical (NBC) protected, track armored amphibious vehicle. The AAAV family consists of a personnel variant (AAAV-P) and a command and control variant (AAAV-C). The first prototype of the AAAV-P will be completed in June of 1999. The vehicle can carry 17-18 combat equipped Marines, in addition to 3 crew members, and weighs 76,000 lbs. when fully loaded. Dimensions of the vehicle are 336" in length, 144" in width, and 120" in height. Figure 1 is an illustration of the AAAV-P.

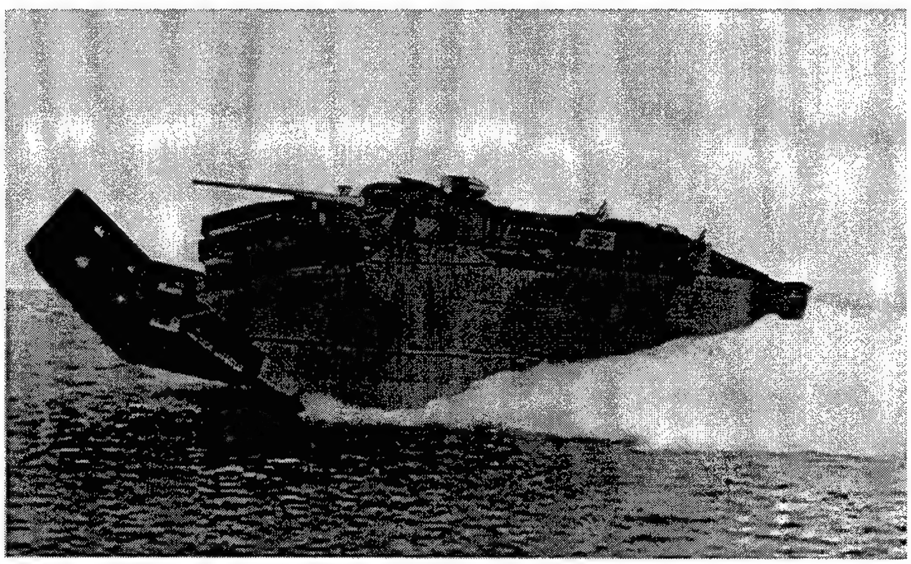

Figure 1. AAAV-P. [1].

The AAAV-P will provide the United States Marine Corps with the capability to maneuver at speeds of 20-25 knots in the water and up to 45 mph on land. The AAAV-P is designed to allow high-speed, seamless maneuver from positions aboard ships to shore locations. The vehicle is powered by a single 2600+ horsepower diesel engine. It carries a 30mm weapons

station/turret with a 7.62mm machine gun. [1] [3] Table 1 provides the technical characteristics of the vehicle.

AAAV Technical Characteristics			
Armored Protection	14.5mm AP @ 300 meters 155/152mm Artillery Fragments @ 50 feet	Roll Recovery	100 degrees
Fire Suppression	Automatic Fire Extinguishing System	Reserve Buoyancy	30%
Suspension	Retractable Hydropneumatic	Ground Clearance	16 inches
Engine	Common Engine Bay	GVW - Fully Loaded	71,000 lbs.
Water Propulsion	Two 23" Diameter Waterjets	GVW - Empty	62,000 lbs.
Primary Weapon	30mm Bushmaster	Ground Pressure @ GVW (21" Track Width)	8.7 psi
Secondary Weapon	7.62mm Machine Gun	Crew	3
Operating Range - Land	400 miles	Combat Equipped Marines	17-18
Operating Range - Sea	75 miles	Cargo Capacity (in lieu of Marines)	5,000 lbs.
Speed - Land	45 MPH	Ammo - 25mm Ready Rounds	300
Speed - High Water Speed Mode	23-29 MPH (20-25 knots)	Ammo - 7.62 Ready Rounds	800
Speed - Transition Mode	8-10 MPH (6-9 knots)	Ammo - 25mm Stowed	600
		Ammo - 7.62mm Stowed	1,600

Table 1. AAAV Technical Characteristics. [4].

The AAAV-P has various modes of operation: transition mode (used to enter and exit the water from ship or land), high-speed water mode, land mode, and silent watch mode (used for a stationary vehicle to reduce acoustic and thermal signature). The communications equipment inside the AAAV-P includes three very high frequency (VHF) Single Channel Ground Airborne Radios System (SINCGARS) radios, a multi-band radio, a Data Automated Communications Terminal (DACT) interface, and a wireless intercom

headset. The Navigation equipment includes an enhanced position location reporting system (EPLRS) and a Global Positioning System (GPS). [1] [3]

A. AAAV-P VETRONICS SYSTEM

Communications within the AAAV-P are highly dependent upon the Vetronics System. The Vetronics System provides the interface among the AAAV-P subsystems. The subsystems contained in the AAAV-P are the Communication, Navigation, Armament, Fire Control, Automotive Drivetrain, Auxiliary Power Unit, Engine, Hydraulics, Hydro System/Appendages, Marine Drive, Suspension, Bilge Pumps, Environmental Control, Fire Detection/Suppression, Hull, NBC, Turret, Electrical Power Management, and Processors/Controls/Displays subsystems.

Figure 2 shows the Vetronics System physical architecture design. The AAAV-P components supported by the Vetronics System are the Command and Control Server (CCS); Control and Display Panel - Driver; Control and Display Panel - Gunner; Control and Display Panel - Troop Commander; Control and Display Panel - Vehicle Commander; Control Handle Assembly - Gunner; Control Handle Assembly - Vehicle Commander; Embedded Training Server; Hull Excitation Reference Source; Hull Electronics Unit (HEU); Hull Power Distribution Unit (HPDU); Mass Memory Unit (MMU); Remote Acquisition Control Modules (RACMs); Turret Electronics Unit (TEU); and the Weapons Station Electronics Unit (WSEU). [5]

The Vetronics System architecture supports three networks within the AAAV-P. The High Speed Data Bus is considered the Main Data Bus. The Utility Bus is the Power Management Bus, while the Powertrain bus is the Automotive Bus. [5] The purpose of the High Speed Data Bus is to support integration of the HEU, TEU, WSEU, and CCS. The

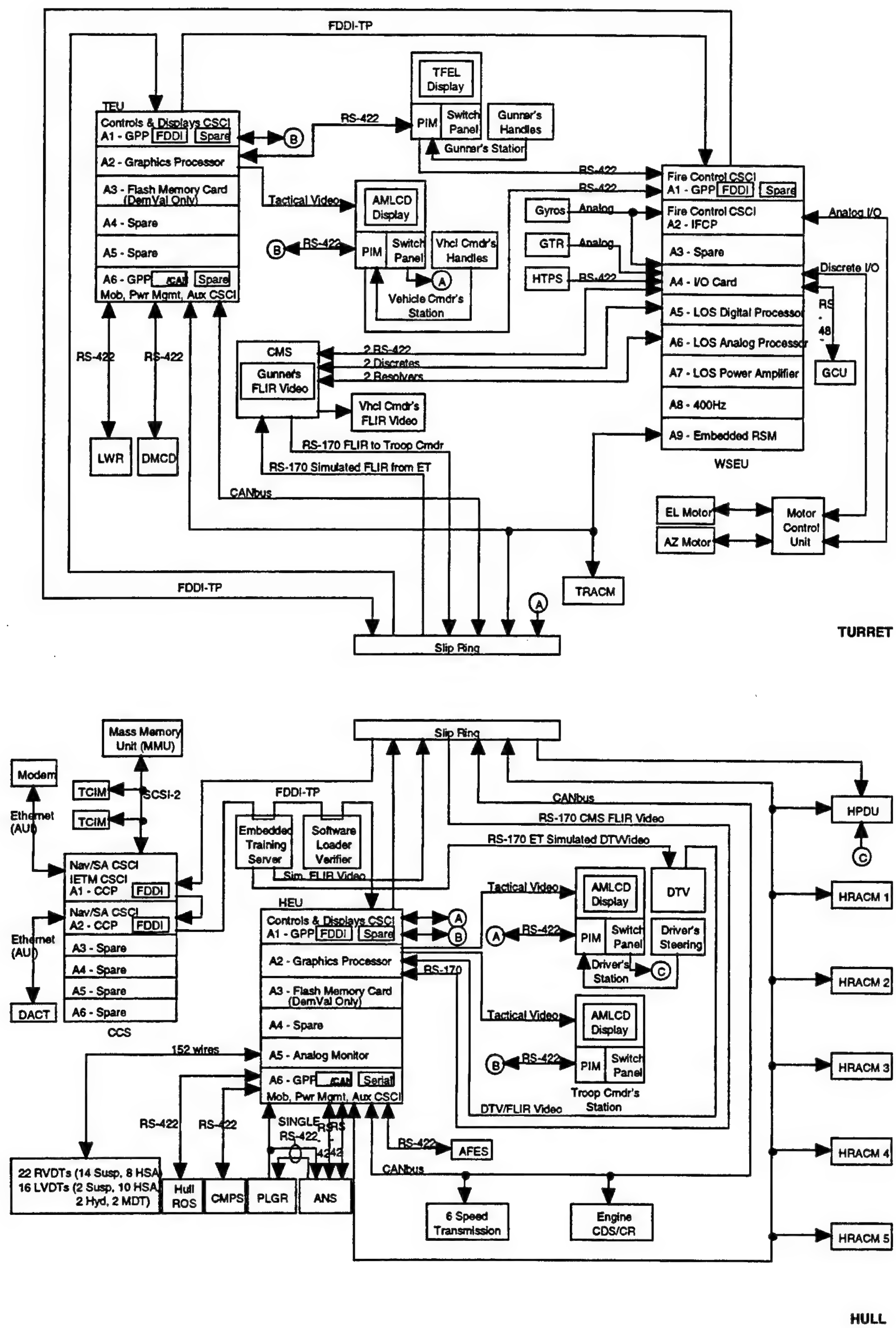

Figure 2. AAAV-P Vetronics System Architecture. [5].

purpose of the Utility Bus is to provide communications between the TEU, the HEU, and the RACMs. The purpose of the Powertrain Bus is to provide communications between the TEU, the HEU, the engine, and the transmission.

As stated previously, the focus of this thesis is on the performance analysis of the High Speed Data Bus. There are four primary nodes on the High Speed Data Bus: TEU, WSEU, HEU, and CCS. The Embedded Training Server and the Software Loader/Verifier are not operational components and are not included in this modeling and simulation effort. The HEU is the primary High Speed Data Bus controller and the TEU is the backup bus controller. The data files of the TEU Controls and Displays (CD) Computer Software Configuration Item (CSCI) must mirror those of the HEU CD CSCI. Similarly, the TEU Mobility/Power Management/Auxiliary (MPA) CSCI data files must mirror those of the HEU MPA CSCI. Therefore, the TEU must be continually updated with message traffic from the HEU.

The CCS provides command and control processing for the vehicle. The CCS hosts the Joint Maritime Command Information System (JMCIS) Command and Control software. The Navigation and Situational Awareness (NAV/SA) CSCI of the CCS receives incoming operational messages via the DACT and the modem. The CCS is responsible for forwarding these operational messages to the HEU, TEU, and WSEU, via the High Speed Data Bus, as required. The NAV/SA also manages the positioning, navigation, and map control capabilities of the AAAV. The MMU provides persistent storage for the map files. The CCS accesses and parses these map files to the TEU, HEU, and WSEU, via the High Speed Data Bus, as required. [5] This is the single largest bandwidth requirement for the High Speed Data Bus.

The HEU provides the processing support for mobility, auxiliary subsystems, and power management functionality. The HEU General Purpose Processor (GPP) module is the bridge between the High Speed Data Bus, Utility Bus, and Powertrain Bus within the Vetronics System communications network. The CD CSCI manages the tactical control and display panel controls and indicators. It also manages the crew handle interfaces for the Vehicle Commander and the Gunner. The MPA CSCI supports the control of the suspension, hydraulics, drivetrain, and engine. [5]

The TEU provides backup processing support for the mobility, auxiliary subsystems, and power management functionality of the HEU and a backup network bridge support between the three buses. [5]

The WSEU supports the vehicle's firepower functionality. It provides torque command to the dual axis motor controller and supports control of the vehicle weapon operations. [5]

The AAAV-P has four terminals, one terminal for each of the three crewmembers and an additional terminal for the troop commander. Each station consists of a color Control Display Panel (CDP) and provides information to the operator in the form of text messages and graphics. [5] Note that the Vehicle Commander's Station and the Gunner's Station are directly linked to the TEU Graphics Processor. Similarly, the Driver's Station and the Troop Commander's Station are directly linked to the HEU Graphics Processor. The vehicle's crew and the Vehicle Commander are dependent upon the accurate operation of the Vetronics System to communicate within the vehicle as well as external to the vehicle.

This thesis is not concerned with all of the components and subsystems listed above. However, they are all mentioned to emphasize the importance of the Vetronics System with respect to the AAAV-P functionality.

B. HIGH SPEED DATA BUS

1. High Speed Data Bus Physical Architecture

This thesis involves the modeling and simulation of the High Speed Data Bus as depicted in Figure 3. The network consists of four nodes: the HEU, TEU, WSEU, and the CCS.

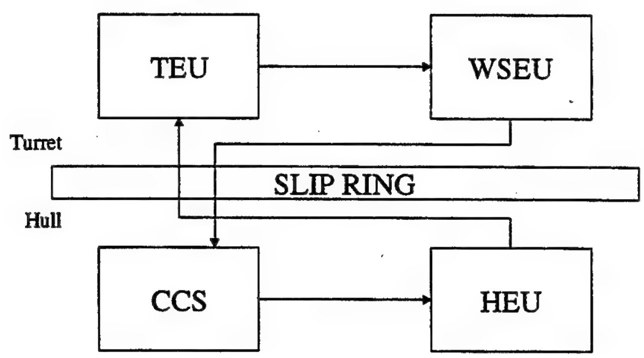

Figure 3. AAAV-P High Speed Data Bus Network Topology. [6].

The nodes are fixed with approximately 2 meters distance between adjacent nodes. The High Speed Data Bus is a dual, counter-rotating, fiber distributed data interface (FDDI) ring. The use of a dual-ring configuration provides system fault tolerance. For the purposes of this thesis, fault tolerance will not be addressed and only a single FDDI ring is modeled.

2. High Speed Data Bus Protocol

a. TCP/IP over a FDDI MAC

Stations on the AAAV-P High Speed Data Bus communicate using TCP/IP over a FDDI Medium Access Control (MAC). TCP is a connection-oriented, end-to-end transport protocol. TCP provides a reliable, ordered packet delivery service using

acknowledgements, sequence numbers, and a sliding window algorithm. TCP uses a three-way handshake to establish connections between clients and servers. First, the client sends a message with a sequence number (x) to the server and identifies it as a request for connection by setting the SYN flag. The server responds to the request with an ACK message, which acknowledges that sequence x+1 is the next expected message, a SYN, and its own sequencing number (y). The client then responds with another ACK message (y+1). [7] Each server that receives a message responds to the originator of the message with an ACK message. This process produces additional overhead on the network. The three-way handshake and message acknowledgement processes are depicted in Figure 4.

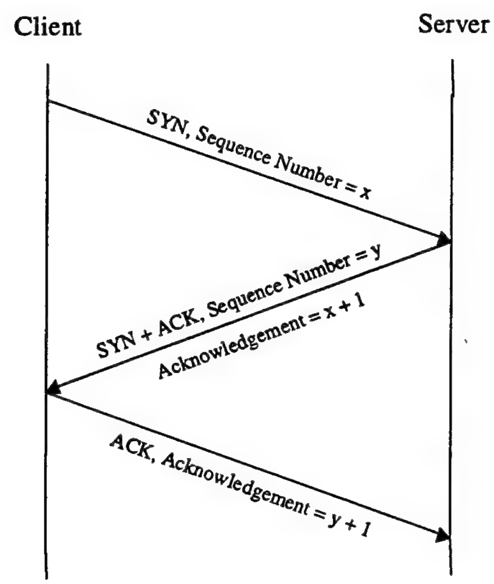

Figure 4. TCP Three-way Handshake Connection Process. [7].

In contrast to TCP, IP is a connectionless, best effort service. IP makes no effort to recover lost packets, but it supports retransmissions. However, the simplicity of IP allows it to be used over dissimilar network types. IP provides a 32-bit global addressing scheme to identify all of the hosts on the network or across multiple networks (internet). IP addresses are written as four integers separated by decimal points. The IP addresses of the four nodes

on the High Speed Data Bus network are shown in Table 2. IP uses the Address Resolution Protocol (ARP) that enables each node on the network to build a table (cache) of mappings between IP addresses and link-level (i.e., physical) addresses.

Host Name	AAAV High Speed Data Bus IP Address
CCS	205.205.205.50
HEU	205.205.205.30
TEU	205.205.205.20
WSEU	205.205.205.40

Table 2. AAAV High Speed Data Bus IP Address Assignments.

The internal network architecture of TCP/IP over a FDDI MAC is shown in Figure 5 as a layered protocol stack. Each layer of the architecture has specific functions to perform. Starting at the top of the stack, the application layer defines the data syntax, which allows applications to interface with each other. Application layer protocols include file transfer, electronic mail, and electronic database protocols. The transport layer provides for reliable transfer of data between stations on the network. It defines the quality of service required and the end-to-end integrity of the network. This is the transmission control protocol (TCP) portion of TCP/IP protocol suite in this case. The network layer controls operations within the network such as switching and routing operations. The network layer is the Internet protocol (IP) portion of TCP/IP protocol suite in this case. The data link layer provides flow control, error checks, and link sequencing. The data link layer is divided into two sublayers: the logical link control (LLC) and the medium access control (MAC). The LLC sublayer provides services between the MAC and the network layer, while the MAC sublayer provides a medium access scheme, either cooperative or with contention, provides address recognition, and generates and verifies frame check sequences. FDDI is implemented at the MAC sublayer. The physical layer is at the bottom of the stack and handles the transmission of the bits across the communications link while controlling the timing and encoding. [6]

LAYER	RESPONSIBILITIES
Application Layer	•Supports protocols to allow applications to function.
Transport Layer	•Provides reliable data transport from the source machine to the destination machine.
Network Layer	•Controls the operation of the subnetwork. •Determines how packets are routed from source to destination.
Data Link Layer	•Provides well-defined service interface to the Network Layer. •Determines how the bits to/from the Physical layer are grouped into frames. •Regulates the flow of frames. •Deals with transmission errors.
Physical Layer	•Transmits raw bits from one machine to another over a physical medium. •Encodes raw bits for transmission as pulses or waveforms.
Physical Medium	•Guided media (e.g., optical fiber).

Figure 5. TCP/IP over FDDI Network Architecture. [6].

b. FDDI Timed Token Ring Access Method

FDDI allows data transfer at 100-Mbps using a timed token ring technique for media access control, which allows all stations to efficiently share the network bandwidth. The token ring technique uses a small frame, called a token, to control access to the network transmission medium. The token is a unique bit pattern that circulates around the ring when all nodes are idle. The token format is shown in Figure 6.

Starting Delimiter (1 Byte)	Frame Control (1 Byte)	Ending Delimiter (1 Byte)

Figure 6. Token Format. [6].

When a station has data to transmit, the station must wait until it detects the token passing by. When the station detects the token, it captures the token and transmits the data. The station can transmit data for a fixed time interval, the token holding time (THT). If all data is transmitted before the THT expires, the token is immediately released after the last frame is transmitted. Once the THT has expired, the transmitting station releases the token back onto the ring. The last frame is allowed to complete if the THT expires in the middle of a frame transmission. Each station on the FDDI network reads the destination address of the packet to determine if it is the recipient of the message. If so, it pulls out the data, and then retransmits the frame to the next station on the ring. The originating station removes the data it transmitted after the data has circulated the ring. It then releases the token if it has not already been released with the trailing edge of the frame. Once the station releases the token back on to the network, the next station that needs to transmit data may do so when it detects the token passing by. The token ring scheme used in FDDI differs slightly from the normal token access method in that the token rotation time (TRT) is limited. The TRT is the time it takes for a token to circulate around the ring. A target token rotation time (TTRT) is agreed upon by all stations. The TTRT shall not exceed 10ms for the AAAV-P High Speed Data Bus. A restricted asynchronous scheme allows stations on the network to restrict the other stations from using the network for asynchronous traffic. The AAAV-P High Speed Data Bus implements non-restricted asynchronous traffic. Furthermore, for non-restricted asynchronous message traffic, the THT for all stations is defined as:

$$THT = TTRT - TRT\,.$$

FDDI allows for the use of up to eight priority levels. However, the AAAV-P implementation does not currently implement priority levels.

c. FDDI Frame Format

FDDI formatted frames are used to transport data packets over the High Speed Data Bus network. The maximum frame size for a FDDI frame is 4500 bytes. The FDDI frame format used for the AAAV-P is shown in Figure 7. Details regarding the content of each portion of the FDDI frame are provided in Appendix B. The FDDI header and trailer provide information such as the destination and source addresses and frame check sequences for bit error detection. The High Speed Data Bus supports individual and broadcast addressing. Addresses are 48 bits in length. The IP header includes information such as time to live and source and destination IP addresses. The TCP header identifies information to include the source and destination ports. [6] The AAAV-P specifications include discussions about employing UDP for synchronous traffic. However, all High Speed Data Bus traffic currently is sent asynchronously via TCP.

FDDI Header (14 Bytes)	IP Header (20 Bytes)	TCP Header (20 Bytes)	AAAV Specific Data (up to 4440 Bytes)	FDDI Trailer (6 Bytes)

Figure 7. FDDI Frame Format. [6].

d. FDDI Performance Measures

Network performance is affected by the number of active stations on the network and the load each station places on the network. Active stations are defined as those stations that are either transmitting or waiting to transmit a frame. Network "throughput" measures the error-free output of the system and is expressed in bits per second (bps) or packets per second (pps). The network "load" measures the input of the system. If the load is less than the network capacity, then the throughput is equal to the load. Network "utilization" measures the level of resource usage, or the percentage of time that the resource is used. The "useable bandwidth" of the network is the maximum

obtainable throughput under high load. The "efficiency" of the network is defined as the ratio of the usable bandwidth to the nominal bandwidth (100 Mbps for FDDI LANs). A FDDI network's productivity is measured by its throughput, while its responsiveness is measured by the response time and access delay. The "response time" is defined as the time between the moment the frame arrives at the queue in order to be transmitted and the time its transmission is complete. The "access delay" is defined as the time between the end of a station's previous transmission and the beginning of a new transmission at the same station. [8]

The efficiency and maximum access delay can be computed using the ring latency (the time required for the bits to circulate the ring) and TTRT values. The ring latency, D, is based on the total fiber length, L, the velocity of signal propagation, v, the number of stations on the ring, s, and the repeater delay, R [8]:

$$D = \frac{L}{v} + s * R$$

For a FDDI network with n active stations, the efficiency and maximum access delay are calculated as follows [8]:

$$Efficiency = \frac{n(TTRT - D)}{n * TTRT + D}$$

$$Maximum\ Access\ Delay = (n-1)TTRT + 2D$$

Additionally, the maximum queue delay can be defined as follows [8]:

$$Maximum\ Queue\ Delay = \frac{Maximum\ Queue\ Length}{Data\ Rate} + Maximum\ Access\ Delay$$

The network load determines which key performance parameters to focus on. When the workload is heavy, network performance is measured by its throughput and access

delay. When the workload is normal, or below saturation, the throughput is equal to the load placed by the active stations, and it therefore not a concern. However, ring latency, response time, and queue delay are useful measures in this situation. [8]

e. FDDI Advantages

There are several advantages to using a FDDI network for the implementation of the AAAV-P High Speed Data Bus. FDDI typically provides a higher bandwidth than other LAN technologies. FDDI also offers a synchronous transmission service. Synchronous transmission is more effective for real-time message traffic since it ensures the messages can get across the network during times of heavy load (currently the AAAV-P only implements asynchronous message traffic). FDDI's dual ring design addresses fault tolerance. FDDI networks have a higher reliability than other LAN technologies because fiber has lower bit error rates than other physical mediums such as copper cable. Additionally, fiber provides a noise-free medium for communications. FDDI is not subject to electromagnetic interference (EMI). [8]

3. High Speed Data Bus Message Traffic

The HEU, TEU, WSEU, and CCS communicate using the High Speed Data Bus. Message traffic concerns a wide range of areas that include the vehicle's operational mode, navigational position, velocity, hatch status, ammunition temperature, lights, and gun position. There are almost 600 different messages that are transmitted among the four nodes on the High Speed Data Bus. The messages are transmitted at various frequencies ranging from 1 Hz to 200 Hz. Message sizes are typically 32 bits. Chapter IV will discuss the simulation associated with this thesis and the traffic load imposed on the High Speed Data Bus.

C. UTILITY BUS

The Vetronics System Utility Bus is a highly reliable, high speed, two-way serial data bus that provides an interface between the TEU, HEU, and RACMs. This bus is based on the bus used within the United States Army's M1A2 Abrams main battle tank. The Utility Bus is implemented to control power management and data acquisition for the AAAV-P. The scope of this thesis is limited to only modeling the impact of the Utility Bus on the High Speed Data Bus (i.e., the traffic load imposed by the Utility Bus). Therefore, the background discussion of the Utility Bus is restricted to providing the reader with a general understanding of the Utility Bus architecture, protocol, and message traffic. This discussion will also identify the impact that the Utility Bus has on the High Speed Data Bus traffic load. For a more detailed description of the Utility Bus, the reader is referred to the AAAV-P Utility Bus documentation [9] [10].

1. Utility Bus Physical Architecture

The HEU is the primary Utility Bus controller, while the TEU is the backup bus controller. The bus controller sends commands to the RACMs. There is one RACM in the AAAV-P Turret, called the Turret RACM (TRACM), and five in the Hull, called the Hull RACMs (HRACMs). The RACMs provide control and sensory data to the bus controller. In addition to these six RACMs, there is an RACM in the Hull Power Distribution Unit (HPDU) that provides electrical loads and signals, and an embedded RACM in the WSEU for electrical load switching. [9] [10] Figure 8 is a depiction of the Utility Bus architecture.

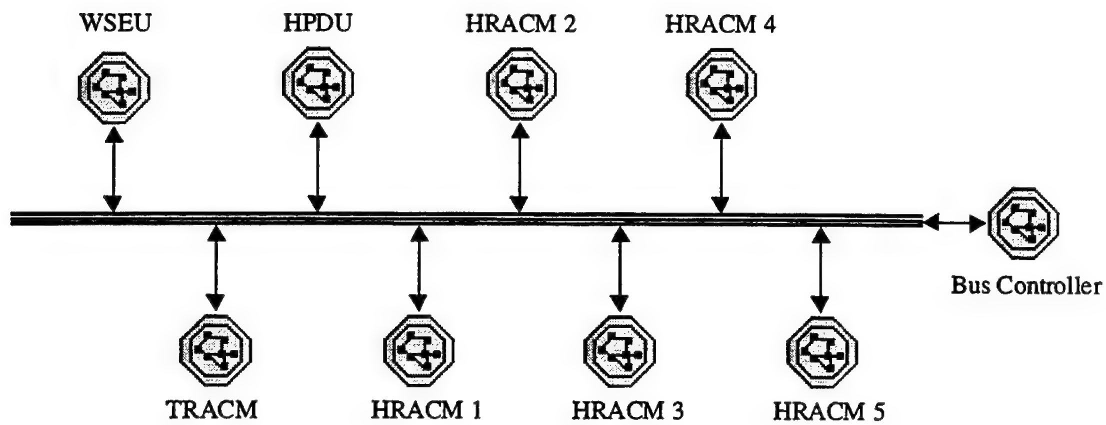

Figure 8. AAAV-P Utility Bus Network Topology.

2. Utility Bus Protocol

The Utility Bus is a deterministic serial data bus that follows a predetermined schedule. The bus is a full command/response protocol in that each command message from the bus controller requires a response from the remote terminal. When the bus controller transmits a command, all remote modules receive the command and evaluate the intended module address. The module with the intended address will immediately respond to the controller to signify a successful receipt. The bus is dual redundant (Bus A and Bus B) which provides for fault tolerance. Additionally, the bus is self-testing. [9]

Six commands are used to communicate on the Utility Bus: *set-up*, *self-test, peek module*, *execute*, *peek single device*, and *peek multiple device*. The *set-up* command is used to transfer initial system set-up data from the bus controller to any of the remote modules. The *self-test* command is used to instruct a remote module to run its self-test routine. The *peek module* command is used to check a remote module's configuration and diagnostic status. The *execute* command is used to instruct a remote module to perform a specific task. The *peek single device* command is used to request the status or data from specific devices assigned to a remote module. The *peek multiple device* command is used to request the

status of data from multiple devices associated with a specific remote module. Additionally, a broadcast command can be issued to instruct the remote module to either turn ON or OFF all associated devices. However, this command does not generate a response from the remote module. [9]

3. Utility Bus Message Traffic

a. Utility Bus Schedule

There are three Utility Bus schedules: Power Up, Operational, and Power Down. The Power Up schedule consists of *execute*, *peek module*, *set up*, *self-test*, and *peek multiple device* commands. The Operational schedule consists of *execute*, *peek module*, *self-test*, *set up*, *peek multiple device*, and *peek single device* commands. The Power Down schedule contains only an *execute* command. The Power Up schedule is generated, based on the power mode selected by the driver or vehicle commander. Once the Power Up schedule is complete, the bus proceeds to the Operational schedule. The Operational schedule runs until the bus controller receives a power down command, at which time the Utility Bus will switch to the Power Down schedule. The Power Down schedule contains only an *execute* command to remove power from the Vetronics system. [10]

b. Utility Bus Impact on the High Speed Data Bus

The Utility Bus will have an impact on the High Speed Data Bus traffic load only when there is a change in the status of the Utility Bus or the devices it is controlling or monitoring. For example, when the contents on the Utility Bus A and Bus B are not equal, High Speed Data Bus messages are generated to identify the discrepancy. Other High Speed Data Bus messages may result in such instances as a failed power down command.

The Utility Bus has proven to be highly reliable. The Utility Bus has been implemented in the United States Army's M1A2 Abrams main battle tank and has demonstrated an estimated 5.03E-04 errors/sec. Recent tests using real hardware in a laboratory environment ran for 12.5 hours with no errors. The result was the transmission of 2.0E11 bits without any errors. [11] Based on these test results, the Utility Bus is expected to have a minimal impact on the High Speed Data Bus traffic load. Chapter IV will discuss the simulation associated with this thesis and the various Utility Bus load scenarios modeled to analyze the impact of the Utility Bus on the High Speed Data Bus.

D. POWERTRAIN (CAN) BUS

In the AAAV-P, the Powertrain Bus, also called the Controller Area Network (CAN) Bus, is used to exchange data between the HEU, the TEU, the engine Control and Diagnostic System (CDS/CR), and the transmission Electronic Control Unit (ECU). The messages transmitted across the CAN Bus are used for engine operation. As in the case of the Utility Bus, the scope of this thesis is limited to modeling the impact of the CAN Bus on the High Speed Data Bus. Therefore, the background discussion of the CAN Bus is restricted to providing the reader with a general understanding of the CAN Bus architecture, protocol, and message traffic. This discussion will also identify the anticipated impact the CAN Bus has on the High Speed Data Bus traffic load. For a more detailed description of the CAN Bus, the reader is referred to the AAAV-P CAN Bus documentation [12] [13].

1. CAN Bus Physical Architecture

The CAN Bus topology is depicted in Figure 9. The CAN Bus wiring topology is as close to linear as possible to avoid cable reflections. The bus line is electrically terminated

at each end with a load resistor of 120 ohms. Furthermore, the nodes are equally spaced on the network to minimize standing waves. [14].

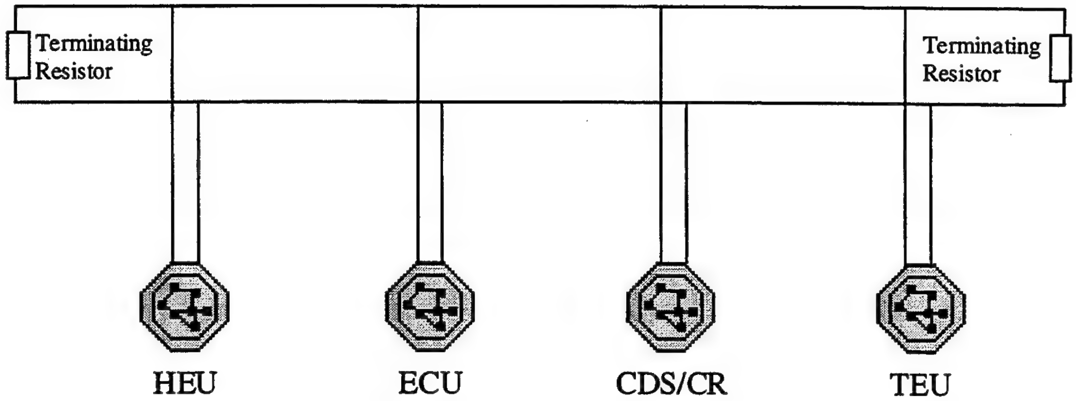

Figure 9. CAN Bus Topology. [12].

2. CAN Bus Protocol

The CAN bus is developed in accordance with SAE specifications 1939/11, 1939/12, 1939/31, 1939/71, and 1939/73. These standards were developed for use in light- and heavy- duty vehicles on- or off- road. The CAN Bus is a serial data link that operates at a data rate of 250 Kbps.

The CAN bus supports five message types: *commands*, *requests*, *broadcast/response*, *acknowledgements*, and *group functions*. A *command* message results in the destination node taking a specific action. A *request* message simply requests information from either one or all nodes. A *broadcast/response* message can be an unsolicited broadcast of information or it can be a response to a command or a request. An *acknowledgement* confirms that a message has been received. A *group function* message is used for groups of special functions required by the CAN Bus implementor.

3. CAN Bus Message Traffic

The messages transmitted across the CAN Bus are used for engine operation. Five different messages are generated on the High Speed Data Bus due to engine-related events that occur on the CAN Bus. These messages concern the transmission status, engine data, engine status, engine malfunctions, and engine faults. The engine data and engine malfunction notification messages potentially place the greatest burden on the High Speed Data Bus as these are transmitted to the TEU 18,000 times an hour, or once every 0.2 seconds. The transmission status message is sent once a minute and the other two messages are sent only five times an hour, therefore having a minimal effect on the High Speed Data Bus message traffic.

III. HIGH SPEED DATA BUS NETWORK MODEL

Optimized Network Engineering Tools (OPNET) is the simulation software used to analyze the AAAV-P Vetronics System for this thesis. Appendix C provides an overview of the OPNET software and its capabilities.

The High Speed Data Bus model developed for this thesis will be discussed in terms of its hierarchy, specifically the network, node, and process levels. Refer to Appendix C and Figure 33 for a depiction of the hierarchical relationship and a discussion of the individual levels.

For a detailed understanding of the models developed for this thesis, it is recommended that the reader proceed in the following manner. Begin by reading this chapter in its entirety. Thereafter, read Appendix C for an overview of the OPNET software. Next, examine the code contained in Appendices D and E. Finally, re-read this chapter.

A. HIGH SPEED DATA BUS NETWORK MODEL

The network shown in Figure 10 is the High Speed Data Bus network model used in this thesis. This model was created using the OPNET Network Editor development tool. The network consists of four nodes: TEU, WSEU, HEU, and CCS. Due to OPNET Version 5.1D limitations, the model employs a central hub to represent a FDDI ring configuration. Although the network uses a central hub, the network is logically and physically a ring topology. The network model shown in Figure 10 mirrors the location of the network nodes in the AAAV-P. However, refer to Figure 3 to see which nodes are adjacent to each other on the network. The nodes are fixed with approximately 2 meters distance between adjacent nodes.

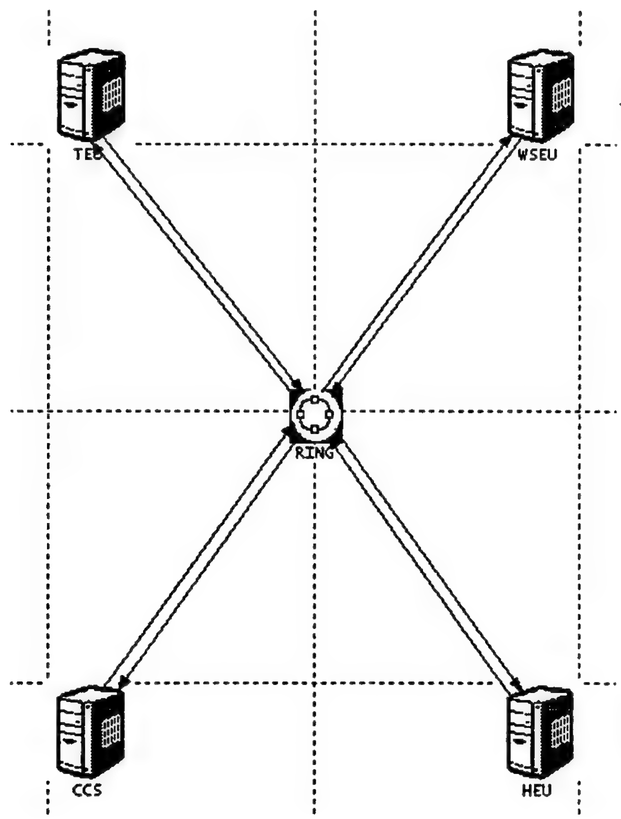

Figure 10. AAAV-P High Speed Data Bus Network Model.

B. HIGH SPEED DATA BUS NODE MODEL

A common node model supports each of the four nodes in the network. This node model, customized for the TEU node, is represented in Figure 11. Figure 11 also depicts the relationship of each node module to the TCP/IP network architecture model discussed in Chapter II and shown in Figure 5.

The node model contains two user-defined processes at the application layer: *teu_msg_rcvr* and *teu_msg_gen*. The other eight processes are available in OPNET's standard library. The node model shown in Figure 11 does not explicitly contain the physical medium layer of the TCP/IP network architecture. Instead, the physical medium is handled at the network node level as shown in Figure 10. The eight FDDI links shown in

Figure 10 represent the physical medium layer. Each layer of the node model is discussed below.

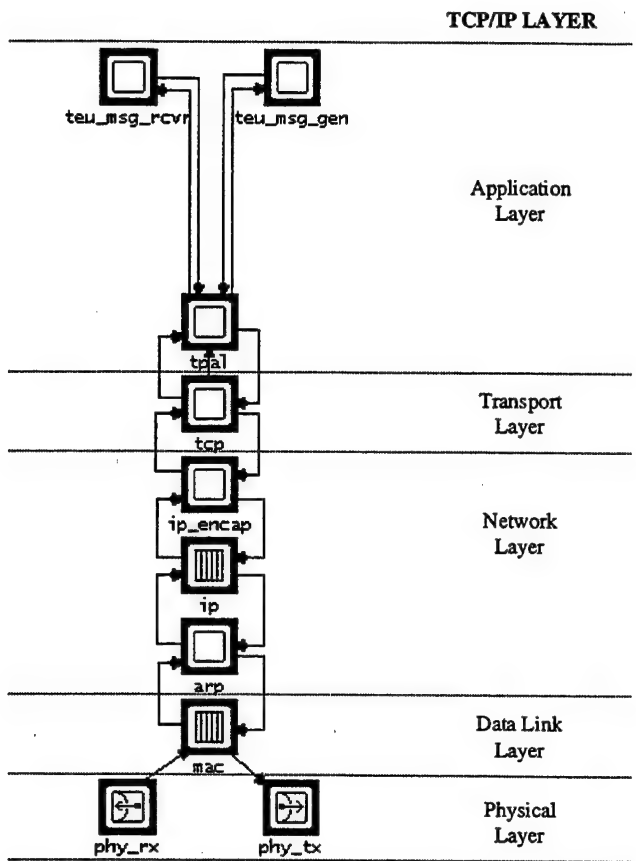

Figure 11. TEU Node Model and TCP/IP Network Architecture Protocol Stack.

1. Physical Layer

The *phy_rx* and *phy_tx* modules are the point-to-point node receiver and transmitter respectively. The *phy_rx* module serves as the inbound interface between communication links outside the node and packet streams inside the node. Similarly, the *phy_tx* module serves as the outbound interface between packet streams inside the node and communication links outside the node. These modules are included in OPNET's standard library and are described further in the OPNET Modeling Volume 1 user's manual [15:Comec].

2. Data Link Layer

The *mac* module represents the data link layer of an FDDI interface. The *mac* module handles transmission requests from the network layer as well as packet arrivals from the physical layer. The *mac* module is also included in OPNET's standard library and is defined by the *fddi_mac_v4* process model. The *mac* module is described further in the OPNET Models/Protocols user's manual [17:FDDI]. The *fddi_mac_v4* process model, contained in OPNET's standard library, is developed so that the MAC of the intended destination of a frame will destroy (strip from the ring) the frame after retrieval rather than repeating it for the originating MAC to destroy it. The models were implemented in this way for efficiency purposes. However, the High Speed Data Bus specification states that the originating station will strip the frame from the network [6]. Therefore, the *mac* process model was modified to reflect this behavior. Specifically, the last section of the INIT state was tailored so that the Boolean efficiency_strip variable is always set to OPC_FALSE.

3. Network Layer

The *arp*, *ip_encap*, and *ip* modules represent the network layer. The *arp* module implements the Address Resolution Protocol (ARP) that maps IP addresses to physical network addresses. The *ip_encap* module provides the interface to the transport layer protocol. The *ip_encap* module encapsulates data packets received from the transport layer into IP datagrams. The *ip* module provides the routing functions, as well as datagram fragmentation and reassembly. All three of these modules are available in OPNET's standard library. The *arp*, *ip_encap*, and *ip* modules are defined by the *ip_arp_v4*, *ip_encap_v4*, *ip_rte_v4* process models respectively, and are described in the OPNET Models/Protocols user's manual [17:IP].

4. Transport Layer

TCP is a connection-oriented transport layer protocol that provides end-to-end reliability using acknowledgments and retransmissions. The *tcp* module performs the responsibilities of the transport layer and is provided in OPNET's standard library. The *tcp* module is defined by the *tcp_manager_v3* process model and is described further in the OPNET Models/Protocols user's manual [17:TCP].

5. Application Layer

The *tpal*, *msg_gen*, and *msg_rcvr* modules represent the application layer of the node. The Transport Adaptation Layer (TPAL) provides the interface between the transport layer and the applications within the node model. The *tpal* module is defined by the *tpal_v3* process model and described in the OPNET Models/Protocol's user's manual [17:TPAL]. The OPNET standard library contains this module to provide a uniform interface between applications and different transport protocols.

The *msg_gen* and *msg_rcvr* modules are user-defined modules that generate and receive message traffic, respectively. Each node in the network model has a slightly different rendition of each of these modules, which are named accordingly (e.g., *teu_msg_gen*, *teu_msg_rcvr*, *heu_msg_gen*, *heu_msg_rcvr*, etc.). These process models are discussed below.

C. HIGH SPEED DATA BUS PROCESS MODELS

The underlying process models for the modules representing the physical, data link, network, and transport layers are provided in OPNET's standard library. These process models are documented in the OPNET Models/Protocols users manual [17] and are not discussed in this thesis. The user-defined *msg_gen* and *msg_rcvr* modules and associated

process models were developed in support of this thesis and will be discussed in detail. This discussion will also include details regarding the relationship between these two modules and OPNET's standard *tpal* module. The *msg_gen* and *msg_rcvr* process models were developed using OPNET's Process Editor development tool. Appendices D and E contain the OPNET code associated with the *msg_gen* and *msg_rcvr* process models respectively.

1. Message Receiver Process Model

A message receiver module (*msg_rcvr*) was developed to support the High Speed Data Bus network model. The process model is designed to receive and process all messages intended for the associated node. All four nodes on the network use the identical process model, which is shown in Figure 12.

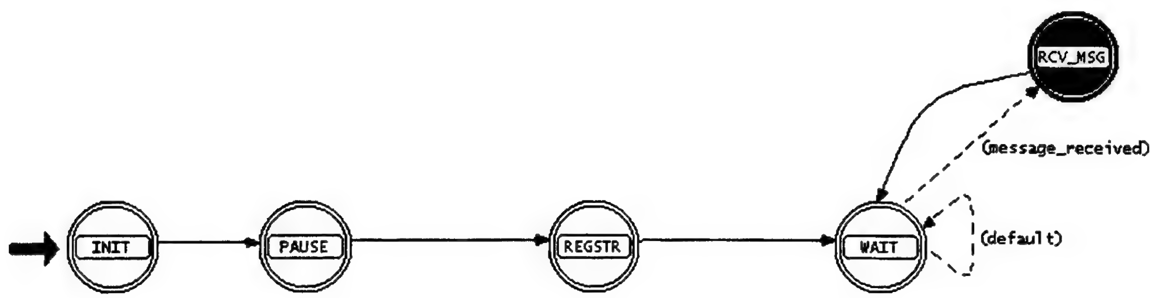

Figure 12. Process Model for the *msg_rcvr* Modules.

The INIT and PAUSE states are unforced states and initialize self interrupts in their enter executives. These self interrupts are needed to return control of the simulation to the Simulation Kernel (SK). Returning control to the SK allows all other processes to be initialized before proceeding further into this process model.

The purpose of the REGSTR state is to register the node as a server on the TCP network. The REGSTR state issues a remote interrupt to provide the TPAL with the

necessary information. A *tpal_serv_reg* ICI is created and installed. Table 3 shows the format of this ICI along with the attribute values used in this modeling effort. The ICI identifies the transport protocol, the name of the service, the local port index, and the popularity value. The transport protocol is always TCP for the current configuration of the High Speed Data Bus network. The service name is user-defined, and is used for debugging purposes only. The port index identifies the port number used by the service. Four ports are established with port indices of 1, 2, 3, and 4. Normally in a TCP implementation, the first 1024 ports are reserved for passive server ports and all ports above 1024 are used for active client ports. However, OPNET does not require models to conform to the port numbering rules governing TCP. The popularity value is a "selection weight" and indicates how often a server is used. This parameter is primarily used when destinations are randomly chosen and is not applicable to this network model. Therefore, the popularity value is 1.0 for all *msg_rcvr* modules. TPAL provides a global registry of all servers registered. [17:TPAL]

tpal_serv_reg		
Attribute Name	**Type**	**Value**
Protocol	structure	tcp
Service Name	structure	FDDI Application - TCP
Port	integer	1 through 4
Popularity	double	1.0

Table 3. *tpal_serv_reg* ICI Format and Attribute Values.

Once the *msg_rcvr* module is registered, the REGSTR state issues a second remote interrupt to the TPAL. This allows the module to establish itself as a server port and to "listen" on the TCP connection for messages intended for the port. A *tpal_req* ICI is created and installed. Table 4 shows the format of this ICI along with the attribute values used in this modeling effort. The *tpal_req* ICI identifies the transport protocol and session. The ICI also contains addressing information and command options. The *msg_rcvr* module is identified as a

passive receiver in the "flags" field. As a passive port, the application reserves local system resources to prepare for incoming requests. The Remote Address and Remote Port are unspecified to enable the message receiver to receive messages from all network nodes and message generators. The Service is user-defined and is utilized for debugging purposes. The Local Ports are numbered 1 through 4 and the protocol is specified as TCP. The other attribute fields are not used. [17]

tpal_req		
Attribute Name	**Type**	**Value**
Transport ID	structure	Not used
Session ID	structure	Not used
flags	integer	TPALC_OPT_PASSIVE
Application ID	structure	Not used
Remote Address	structure	TpalC_Host_Unspec
Service	structure	FDDI Application - TCP
Remote Port	integer	TpalC_Port_Unspec
Local Port	integer	1 through 4
Protocol	structure	tcp

Table 4. *tpal_req* ICI Format and Attribute Values.

The process of issuing these two remote interrupts is performed four times. When each interrupt is executed, a different port index is identified for each connection. A connection is established for each node on the network, plus an additional connection to allow standard numbering of port connections. This will be explained further in the *msg_gen* process discussion. Registration of the servers occurs without a time lapse during the OPNET simulation.

The WAIT state is an idle state that waits for a message to be received. When a packet is received at the intended node, each layer of the node model will handle the incoming packet accordingly until the packet has reached its destination port at the application layer. When the *tcp* module passes the packet to the TPAL, the *tpal* module issues a stream interrupt for the *msg_rcvr* module. When the *msg_rcvr* module receives the

stream interrupt from the SK, the state transition condition "message_received" becomes TRUE and the process proceeds to the RCV_MSG state.

Currently, the RCV_MSG state accepts and then destroys the arriving packet. However, this state could be further developed to process the arriving packet and extract data fields. This would be useful in determining if a response is required to acknowledge receipt of a particular message or to respond to a specific message with additional information. After the message has been destroyed, the process returns to the WAIT state to await the receipt of another stream interrupt from the SK, identifying receipt of another message to be processed.

When the OPNET simulation is complete, the *msg_rcvr* process is destroyed and the process termination block is invoked. The *msg_rcvr* termination block issues a remote interrupt to the *tpal* module to close all TCP connections.

2. Message Generator Process Model

Four message generator modules (*msg_gen*) were developed to support the High Speed Data Bus network model, one for each node on the network. The purpose of the message generator process is to generate the message traffic sent from each node on the network. The generic process model developed for each of the message generator modules is contained in Figure 13. The process model is identical for each network node, except for subtle, but important, differences in the ESTCONN, RD_GDF, and SEND states, as discussed below.

Within the INIT state enter executive, the *act_connect* and *seed* state variables are initialized. The purpose of these state variables will be described later. The INIT state is a forced state. Therefore, the process proceeds immediately to the DELAY state.

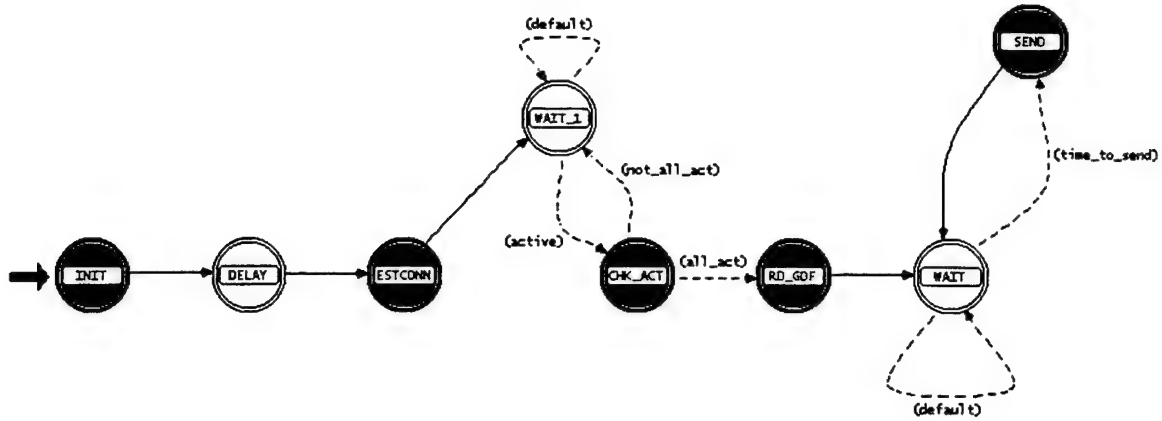

Figure 13. Process Model for the *msg_gen* Modules.

The purpose of the DELAY state is to pass control back to the SK and allow time for all four of the node servers to register themselves. The process passes control to the SK using a self interrupt. All of the servers need to be registered before this process proceeds to the next state, which establishes connections with each of the servers.

The purpose of the ESTCONN state is to establish TCP connections with each of the other network node servers. This is accomplished through the TPAL. The ESTCONN state creates a *tpal_req* ICI and issues a remote interrupt to open a connection with a specified remote port. A separate ICI is created for each connection request. The ICI associated with the remote interrupt contains the necessary information to establish the connection. The *tpal_req* ICI format and the attribute values used for the TEU in this modeling effort are shown in Table 5. The attribute values for the other three network nodes are similar.

The *msg_gen* module is an active port, as specified in the flags field. As an active port, the application connects to a waiting application (i.e., a *msg_rcvr* module) at a designated remote node. The Remote Address corresponds to the Server Address specified in the network model. The Service information is provided for debugging purposes only.

tpal_req		
Attribute Name	**Type**	**Value**
Transport ID	structure	Not used
Session ID	structure	Not used
flags	integer	TPALC_OPT_ACTIVE
Application ID	structure	Not used
Remote Address	structure	WSEU HEU CCS
Service	structure	FDDI - source TEU
Remote Port	integer	1
Local Port	integer	3 (to communicate with the WSEU) 4 (to communicate with the HEU) 5 (to communicate with the CCS)
Protocol	structure	tcp

Table 5. *tpal_req* ICI Format and Attribute Values for the TEU.

The Protocol is always specified as tcp for the current High Speed Data Bus network configuration. The Remote Port identifies the port index at the remote node with which the active connection is established. The numbering is kept constant for each network node to facilitate program coding and debugging. As briefly mentioned in the preceding section, this is the reason the REGSTR process in each *msg_rcvr* module establishes four port indexes for TCP connections. Table 6 summarizes the relationship among the network node modules and the Local Port, Remote Address, and Remote Port assignments. For example, the TEU always establishes a connection to port 1 of each remote node. Similarly, the HEU always establishes a connection to port 2, the WSEU always establishes a connection to port 3, and the CCS always establishes a connection to port 4 of each remote node. This simplifies the port number assignments. However, this leaves four receiving ports unused (i.e., TEU port 1, WSEU port 2, etc.). Figure 14 graphically displays the network nodes and the Local Port, Remote Address, and Remote Port assignments.

module	Local Port Index	Remote Address	Remote Port Index
teu_msg_gen	5	WSEU	1
teu_msg_gen	6	HEU	1
teu_msg_gen	7	CCS	1
wseu_msg_gen	5	TEU	2
wseu_msg_gen	6	HEU	2
wseu_msg_gen	7	CCS	2
heu_msg_gen	5	WSEU	3
heu_msg_gen	6	TEU	3
heu_msg_gen	7	CCS	3
ccs_msg_gen	5	WSEU	4
ccs_msg_gen	6	HEU	4
ccs_msg_gen	7	TEU	4

Table 6. Summary of Local Port, Remote Address, and Remote Index Assignments.

Each time the *tpal* module receives a remote interrupt, it initiates the three-part handshaking sequence that is necessary to open a TCP connection. As an example, the TEU will send an SYN request to each of the WSEU, HEU, and CCS nodes to request to establish TCP connections. The SYN message contains connection initialization information. In return, each of these remote nodes will respond with an SYN ACK (acknowledgment) message to the TEU. When the TEU receives an SYN ACK message from a remote node, the connection is established, the *tpal* module is ready to receive, and an ACK is sent from the TEU to the acknowledging remote node. Once this acknowledgment is received at the remote node, the three-part handshaking sequence is complete. The process of establishing all 12 connections requires approximately 3.6 msec (real time) during the OPNET simulation.

The ESTCONN state is a forced state. After all remote interrupts are issued requesting to establish TCP connections with the other three remote nodes, the process proceeds to the WAIT_1 state. The transition condition "active" becomes TRUE when the process receives a remote interrupt from the SK confirming an open connection. Each time

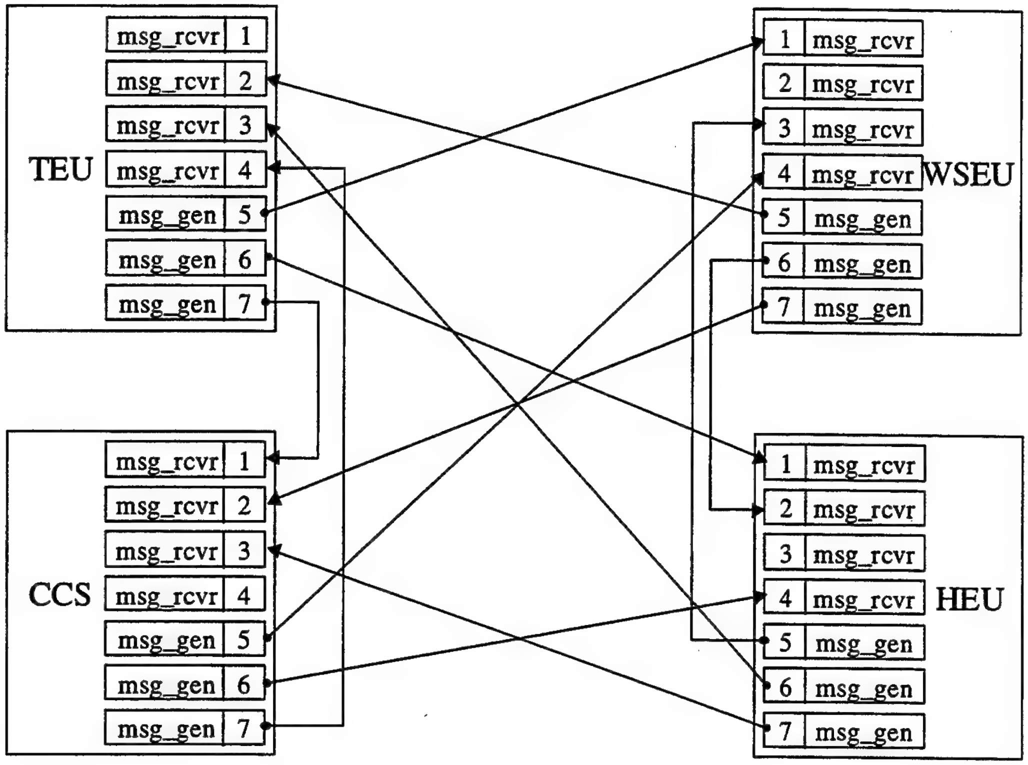

Figure 14. Local Port, Remote Address, and Remote Port Assignments.

this event occurs, the process transitions to the CHK_ACT state which increments the *act_connect* state variable. The *act_connect* variable serves as a counter to count the number of active connections established. If *act_connect* is less than three, the "not_all_act" transition condition is TRUE and the process transitions back to the WAIT_1 state. After all three connections are established, *act_connect* will equal three and the "all_act" transition condition becomes TRUE. The process then proceeds to the RD_GDF state.

The purpose of the RD_GDF state is to identify the message traffic to be sent from each node and to schedule the initial self interrupts required to send the messages. The RD_GDF state begins by reading the general data file (GDF) specified within the RD_GDF state. Each node reads a different file corresponding to that network node and the current scenario. For example, the “fddi_heu_s1” GDF is read by the *heu_msg_gen* process during

scenario 1. The scenarios are discussed in detail in the following chapter and the GDFs are provided in Appendix C.

Each line of the GDFs contains a list of six comma-delineated fields for each message to be sent:

1) message name - the name of the High Speed Data Bus message.

2) destination - the node the message is intended for (HEU, TEU, WSEU, or CCS).

3) size of data - the size of the message in bits.

4) frequency - the number of times per hour the message is sent.

5) variance - the sum of the squared differences around the arithmetic mean divided by the sample size minus one.

6) distribution - normal, constant, or uniform.

Each of these fields is decomposed into a structure, which is defined in the *msg_gen* process header block. A random start time is established for each message based on the *seed* state variable initialized in the INIT state. The purpose of the random start time is to smooth the initial message transmissions over the first 50 seconds of the simulation to avoid an initial spike of throughput at the beginning of the simulation. After all interrupts are scheduled, the process proceeds to the WAIT state in order to wait to receive a self interrupt from the SK.

When a self interrupt is received from the SK, "time_to_send" becomes TRUE and the process transitions to the SEND state. The SEND state creates and sends a packet with the AAAV-FDDI_pk packet format. The message destination determines which ICI is installed before attempting to send the packet.

The AAAV-FDDI_pk packet format is shown in Figure 15. The packet format was defined using OPNET's Packet Format Editor development tool and is in accordance with

the High Speed Data Bus interface control document (ICD). The Packet Update Identification Tag (PUIT) is an eight-bit integer counter used to determine whether the received message contains new information or whether it is superseded by a message previously received. The PUIT and the Data fields are not used in this network simulation. Their existence in the FDDI packet format serves as a placeholder for future modeling efforts. The FDDI packet size is set using the size_of_data field for each message which was extracted during the RD_GDF state. After the packet is sent during the SEND state enter executive, the SEND exit executive is completed. The SEND exit executive schedules another self interrupt to send the message again. The next interrupt is scheduled based on the message distribution, frequency, and variance (if applicable). Determining another send_rate each time the interrupt is scheduled produces a message generation process that is truly random. The process then returns to the WAIT state to await another self interrupt from the SK.

PUIT (8 Bits)	Data (up to 4439 Bytes)

Figure 15. AAAV-FDDI Packet Format.

When the OPNET simulation is complete, the *msg_gen* process is destroyed and the process termination block is invoked. The *msg_gen* termination block destroys all of the ICIs created during the process and issues a remote interrupt to the tpal module to close all TCP connections.

IV. SIMULATION SCENARIOS

To accurately analyze the performance of the High Speed Data Bus, it was necessary to generate scenarios that identify potential performance bottlenecks within the system and determine the network throughput capacity. This chapter identifies and discusses the various scenarios used during the simulation of the model. This chapter also identifies the modeling assumptions and basis for the scenarios developed.

A. MODELING ASSUMPTIONS

Several assumptions were made during the modeling and simulation efforts. These assumptions are listed below.

- For the purposes of this simulation effort, all of the network traffic includes High Speed Data Bus messages identified in the FDDI ICD Appendix A [16] along with the addition of Operational Message Traffic and Map Database Message Traffic.
- All messages are transmitted asynchronously via the TCP protocol. UDP is not employed. FDDI priorities are not implemented and FDDI MAC addresses are 48 bits in length.
- The FDDI TTRT is set at 10 msec in accordance with the FDDI ICD specification [6].
- The Operational Message Traffic is modeled with a message length of 500 bytes and an average transmission frequency of 180 times per hour, distributed normally, for each destination node. This implies that three separate operational messages are sent from the CCS to the TEU, HEU, and WSEU, each with its own frequency and distribution. Operational Message Traffic is not broadcasted.

- The Map Database Message Traffic is modeled with either a message length of 500 Kbytes or 1.0 Mbytes depending on the scenario. The messages have an average transmission frequency of 120 times per hour, distributed normally, for each destination node. This implies that three separate map database messages are sent from the CCS to the TEU, HEU, and WSEU, each with its own frequency and distribution. Map Database Message Traffic is not broadcasted.
- Unless otherwise stated, each message listed in the FDDI ICD [16] is modeled with a size of 32 bits, including the 8-bit PUIT. When an alternate size is specified in the "Notes" column of the spreadsheet, that value is used instead.
- H-CD_TO_H-MPA and H-MPA_TO_H-CD messages are sent to the TEU via the High Speed Data Bus. T-CD_TO_T-MPA and T-MPA_TO_T-CD messages are not sent to the HEU via the High Speed Data Bus.
- All queues are infinite in size. Queue size will be monitored during simulations to determine an adequate size to ensure zero packet loss.
- The distance between adjacent stations on the High Speed Data Bus is modeled as approximately 2 meters.
- No further TCP options are employed in any scenarios.

B. SOURCE DOCUMENTATION

The High Speed Data Bus Interface Control Document (ICD) [6], supplied by the AAAV Project Office, provides the specific messages transmitted among the four nodes on the High Speed Data Bus. Appendix A of the ICD is a Microsoft Excel spreadsheet that identifies, among other information, the individual messages and their associated signal name(s), source CSCI, destination CSCI, and transmission rate.

Figure 43 in Appendix F shows the High Speed Data Bus interfaces for the AAAV-P Vetronics System and reflects the complete set of messages provided in Appendix A of the ICD.

To complete the simulation of the model, it is necessary to model the random inter-departure times for each message using the average transmission frequency, variance, and distribution. The average message transmission frequency ranges between 1 time per hour and 720,000 times per hour. The distributions are assumed either normal or constant, and in accordance with discussions with the AAAV-P Project Office, the variances are assumed to be small. Appendix F contains an excerpt of the Excel spreadsheets provided by the AAAV Project Office. Two columns were added to the Excel spreadsheet, "Frequency (x times hour)" and "Distribution", containing the respective estimates for each message. A column labeled "Source Bus" was also added when applicable. The "Source Bus" column is only applicable when the source CSCI is the HEU MPA. If the High Speed Data Bus message is generated in response to an event occurring on the Utility Bus, the column data is UB. Similarly, if the High Speed Data Bus message is generated in response to an event occurring on the CAN Bus, the column data is CB. Other source buses include NAV422, NAV423, AFES422, and the Analog Monitor Board (AMB). The AAAV Project Office provided guidance and comments throughout the development of these additional columns. [18] [19]

C. HIGH SPEED DATA BUS DATA RATES

Due to AAAV-P hardware limitations, the High Speed Data Bus throughput will never achieve the rated throughput of 100 Mbps typically available in FDDI implementations. Currently, the High Speed Data Bus is expected to operate between 16

and 64 Mbps. Therefore, each scenario described below is simulated at data rates of 16, 32, 48, and 64 Mbps. [20]

D. SCENARIO DESCRIPTIONS

Eleven scenarios were developed to test the performance capability of the High Speed Data Bus. The variations among the scenarios include the volume of message traffic generated as a result of events on the Utility and CAN buses as well as the volume of operational and map database messages traversing the network. The scenarios build upon each other in that each scenario imposes a heavier message traffic load than the previous one. These scenarios were provided to the AAAV Project Office for review [21]. Each of the eleven scenarios is described below.

1. Scenario 1

Scenario 1 message traffic consists of all High Speed Data Bus messages that are not generated in response to events on the Utility and CAN buses. Table 7 contains a listing of the FDDI message traffic for Scenario 1. This group of messages will be referred to as the "Basic FDDI" messages throughout this section.

2. Scenario 2

Scenario 2 message traffic consists of all the Basic FDDI messages identified in Scenario 1 with the addition of those messages that are generated in response to events on the Utility and CAN buses. The High Speed Data Bus messages generated in response to events occurring on the Utility Bus are modeled at a low frequency transmission rate. Each of these messages is generated one time per hour with a normal distribution. The High Speed Data Bus messages generated in response to events occurring on the CAN Bus are modeled at the same frequency transmission rates during each scenario. These messages

and associated frequency transmission rates were described in Chapter II. Table 8 contains a listing of the High Speed Data Bus message traffic for Scenario 2.

Source	Destination
TEU-CD	WSEU-FC
TEU-CD	CCS-NAV/SA
TEU-MPA	CCS-NAV/SA
HEU-CD	TEU-MPA
HEU-CD	TEU-CD
HEU-CD	CCS-NAV/SA
HEU-CD	WSEU-FC
HEU-MPA[1]	TEU-CD
HEU-MPA[2]	WSEU-FC
HEU-MPA	CCS-NAV/SA
WSEU-FC	TEU-CD
WSEU-FC	HEU-CD
WSEU-FC	HEU-MPA
WSEU-FC	CCS-NAV/SA
CCS-NAV/SA	TEU-CD
CCS-NAV/SA	HEU-CD

Table 7. Scenario 1 Message Traffic - Basic FDDI Messages.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[4]	TEU-CD
HEU-MPA[4]	WSEU-FC

Table 8. Scenario 2 Message Traffic.

3. Scenario 3

Scenario 3 message traffic consists of all the Basic FDDI messages identified in Scenario 1 with the addition of those messages that are generated in response to events on the Utility and CAN buses. The High Speed Data Bus messages generated in response to

[1] Messages resulting from the GPS, AMB, NAV422 and ROS422 buses only.

[2] Messages resulting from the NAV422, NAV423, NAV424, and NAV425 buses only.

[3] Messages resulting from events occurring on the CAN Bus.

[4] Messages resulting from events occurring on the Utility Bus - low frequency transmission rate.

events occurring on the Utility Bus are modeled at a higher frequency transmission rate. Transmission frequencies are not uniform across all messages and are identified in Appendix G. Table 9 contains a listing of the High Speed Data Bus message traffic for Scenario 3.

Source	**Destination**
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[5]	TEU-CD
HEU-MPA[5]	WSEU-FC

Table 9. Scenario 3 Message Traffic.

4. Scenario 4

Scenario 4 message traffic consists of all the messages identified in Scenario 2 with the addition of Operational Message Traffic. The operational messages are received by the CCS-NAV/SA via the DACT or modem and are forwarded to the TEU-CD, HEU-CD, and WSEU-CD via the High Speed Data Bus. The size of the messages is modeled as 500 bytes, and the average transmission frequency is estimated at 180 times per hour with a normal distribution. Assumptions regarding the operational message size and frequency were provided to the AAAV Project Office for review [22]. Table 10 contains a listing of the High Speed Data Bus message traffic for Scenario 4.

Source	**Destination**
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[4]	TEU-CD
HEU-MPA[4]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD

Table 10. Scenario 4 Message Traffic.

[5] Messages resulting from events occurring on the Utility Bus - high frequency transmission rate.

[6] Operational Message Traffic.

5. Scenario 5

Scenario 5 message traffic is identical to that of Scenario 4 except that the High Speed Data Bus messages generated in response to events occurring on the Utility Bus are modeled at a high frequency transmission rate. Table 11 contains a listing of the High Speed Data Bus message traffic for Scenario 5.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[5]	TEU-CD
HEU-MPA[5]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD

Table 11. Scenario 5 Message Traffic.

6. Scenario 6

Scenario 6 message traffic is identical to that of Scenario 4 with the addition of map database messages that are sent from the CCS to the TEU, HEU, and WSEU via the High Speed Data Bus. Map database messages are transmitted from the CCS-NAV/SA to the TEU-CD, HEU-CD, and WSEU-CD. The map database message size is estimated to be relatively small, 500 Kbytes, and the average transmission frequency is estimated at 120 times per hour with a normal distribution. The assumptions regarding the map database message size and frequency were provided to the AAAV Project Office for review [22]. Table 12 contains a listing of the High Speed Data Bus message traffic for Scenario 6.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[4]	TEU-CD
HEU-MPA[4]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD
CCS-NAV/SA[7]	TEU-CD
CCS-NAV/SA[7]	HEU-CD
CCS-NAV/SA[7]	WSEU-CD

Table 12. Scenario 6 Message Traffic.

7. Scenario 7

Scenario 7 message traffic is identical to that of Scenario 6 except that the messages generated in response to events on the Utility bus are modeled at a high frequency transmission rate. Table 13 contains a listing of the High Speed Data Bus message traffic for Scenario 7.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[5]	TEU-CD
HEU-MPA[5]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD
CCS-NAV/SA[7]	TEU-CD
CCS-NAV/SA[7]	HEU-CD
CCS-NAV/SA[7]	WSEU-CD

Table 13. Scenario 7 Message Traffic.

8. Scenario 8

Scenario 8 message traffic is identical to that of Scenario 6 except map database messages are assumed to be larger, 1.0 Mbytes, but with the same transmission

[7] Map Database Message Traffic - small message size.

characteristics. Table 14 contains a listing of the High Speed Data Bus message traffic for Scenario 8.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[4]	TEU-CD
HEU-MPA[4]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD
CCS-NAV/SA[8]	TEU-CD
CCS-NAV/SA[8]	HEU-CD
CCS-NAV/SA[8]	WSEU-CD

Table 14. Scenario 8 Message Traffic.

9. Scenario 9

Scenario 9 message traffic is identical to that of Scenario 8 except that the messages generated in response to events on the Utility bus are modeled at a high frequency transmission rate. Table 15 contains a listing of the High Speed Data Bus message traffic for Scenario 9.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[5]	TEU-CD
HEU-MPA[5]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD
CCS-NAV/SA[8]	TEU-CD
CCS-NAV/SA[8]	HEU-CD
CCS-NAV/SA[8]	WSEU-CD

Table 15. Scenario 9 Message Traffic.

[8] Map Database Message Traffic - large message size.

10. Scenario 10

Scenario 10 message traffic is identical to that of Scenario 7 with the addition of Battlesight and Boresight related messages transmitted by the WSEU-FC to the TEU-CD. Battlesight and Boresight messages are modeled at a constant transmission rate of 60 messages every 5 minutes [19]. Table 16 contains a listing of the High Speed Data Bus message traffic for Scenario 10.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[5]	TEU-CD
HEU-MPA[5]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD
CCS-NAV/SA[7]	TEU-CD
CCS-NAV/SA[7]	HEU-CD
CCS-NAV/SA[7]	WSEU-CD
WSEU-FC[9]	TEU-CD

Table 16. Scenario 10 Message Traffic.

11. Scenario 11

Scenario 11 message traffic is identical to that of Scenario 10 except that the map database messages are modeled using the large size of 1.0 Mbytes. Table 17 contains a listing of the High Speed Data Bus message traffic for Scenario 10.

E. GENERAL DATA FILES

The OPNET model developed for this research effort was designed to read ASCII script files to obtain the message traffic parameters for each network node. The ASCII general data files (GDFs) written for this model are listed in Appendix E. These GDFs are a subset of the information listed in the Excel spreadsheets mentioned previously. The GDFs

are read by the RD_GDF state within the *msg_gen* processes. Each process reads a different GDF depending on the network node and the scenario currently being simulated. Table 18 identifies the nodes and associated GDF read for each scenario. Each of these GDFs is listed in Appendix G.

Source	Destination
Basic FDDI messages	
HEU-MPA[3]	TEU-CD
HEU-MPA[5]	TEU-CD
HEU-MPA[5]	WSEU-FC
CCS-NAV/SA[6]	TEU-CD
CCS-NAV/SA[6]	HEU-CD
CCS-NAV/SA[6]	WSEU-CD
CCS-NAV/SA[8]	TEU-CD
CCS-NAV/SA[8]	HEU-CD
CCS-NAV/SA[8]	WSEU-CD
WSEU-FC	TEU-CD

Table 17. Scenario 11 Message Traffic.

	Network Node and General Data File			
Scenario	**TEU**	**WSEU**	**HEU**	**CCS**
1	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s1	fddi_ccs_s1-3
2	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s1-3
3	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s1-3
4	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s4-5
5	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s4-5
6	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s6-7-10
7	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s6-7-10
8	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s8-9-11
9	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s8-9-11
10	fddi_teu_all	fddi_wseu_s10-11	fddi_heu_s2-10-even	fddi_ccs_s6-7-10
11	fddi_teu_all	fddi_wseu_s10-11	fddi_heu_s3-11-odd	fddi_ccs_s8-9-11

Table 18. Network Nodes, Scenarios, and Associated General Data File.

[9] Battlesight and Boresight related message traffic.

V. SIMULATION RESULTS

The goal of this thesis is to analyze the performance capability of the AAAV-P Vetronics System High Speed Data Bus. As discussed in Chapter II, network performance is affected by the number of active stations on the network and the load each node places on the network. The relative importance of different performance metrics depends on the network load. When the network load is low (far below saturation), response time and queue delays are important. When the network load is high (close to or above the link capacity), throughput becomes a more important metric to gauge network performance. The OPNET simulation results with respect to these performance metrics are discussed further below.

A. SIMULATION TECHNIQUE

The OPNET simulation tool can be used to extract a large range of statistics. By using "statistics probes," OPNET can collect statistics on the overall network, a node, a module, or a link. Objects that generate statistics include processors, queues, transmitter channels, receiver channels, and links [15]. Four statistics are collected to support the analysis of the High Speed Data Bus. These include the OPNET-defined link throughput and MAC queue size statistics. Additionally, two user-defined statistics are collected to measure the arrival rate of specific messages. These four statistics are defined as follows:

- <u>Link throughput (bits/sec)</u> - represents the average number of bits successfully received or transmitted by the receiver or transmitter channel per unit time.
- <u>MAC queue size (bits)</u> - represents the current number of bits in the queue awaiting transmission.

- Pitch Angle Rate (sec^{-1}) - represents the arrival rate of the PITCH_ANGLE_FDDI messages at the WSEU.
- Roll Angle Rate (sec^{-1}) - represents the arrival rate of the ROLL_ANGLE_FDDI messages at the WSEU.

The link throughput statistic is used to determine the required bandwidth necessary to support (i.e., avoid packet loss and minimize message delay jitter) the anticipated message traffic on the AAAV-P High Speed Data Bus. The maximum MAC queue size is used to determine an adequate queue size for the AAAV-P hardware to prevent overflow. The user-defined Pitch Angle Rate and the Roll Angle Rate statistics are analyzed to determine the frequency of the associated arriving messages and to determine if the delay jitter is operationally acceptable. The PITCH_ANGLE_FDDI and ROLL_ANGLE_FDDI messages are selected because they have the highest frequency of transmission and are the most sensitive to network load.

The TEU, HEU, and WSEU employ the same hardware suite, while the CCS employs a different hardware suite. The hardware specifications are identified in [5]. Therefore, the queue size measurements are reported as the maximum of the TEU, HEU, and WSEU simulation results, while separate queue size measurements are collected and reported for the CCS.

The OPNET software exhibits some anomalies with respect to the simulation results. These are mentioned briefly throughout the discussion of the simulation results. They are also discussed at the end of this chapter. However, the results observed during the various simulations are viable and are considered to realistically reflect the data throughput placed on the network as well as indicate the queue size requirements.

B. NETWORK PERFORMANCE ANALYSIS

Each of the eleven scenarios described in Chapter IV was simulated for a duration of one hour (3600 seconds) with link data rate capacities of 16, 32, 48, and 64 Mbps. The results are discussed below. Table 24 and Table 25, located at the end of this chapter, summarize all of the simulation results.

1. High Speed Data Bus Network Throughput

Chapter IV and Appendix G identify the various message traffic loads imposed during Scenarios 1, 2, and 3. These scenarios include the Basic FDDI messages and the messages resulting from occurrences on the CAN and Utility Buses at low and high loads. As expected, the network throughput during each of these scenarios is steady, and the 16Mbps network is unsaturated. Figure 16 displays the simulation network throughput results for Scenario 1. The maximum throughput is measured to be 679.3Kbps.

The initial rise in throughput is due to the TCP handshaking process that occurs at the beginning of the simulation. Additionally, all message traffic start times are smoothed over the first 50 seconds of the simulation, causing the second sudden rise. This rapid increase in throughput can be seen in Figure 16.

Figure 17 and Figure 18 display the simulated network throughputs for Scenarios 2 and 3, respectively. The maximum throughput measured for each of these scenarios is presented in Table 19.

	Scenario		
	(1)	(2)	(3)
Maximum Link Capacity	**Maximum Observed Throughput (bits/sec)**		
16 Mbits/sec	679.3K	704.6K	740.5K
32 Mbits/sec	679.3K	704.6K	740.5K
48 Mbits/sec	679.3K	704.6K	740.5K
64 Mbits/sec	679.3K	704.6K	740.5K

Table 19. Summary of Data Throughput Simulation Results for Scenarios 1, 2, and 3.

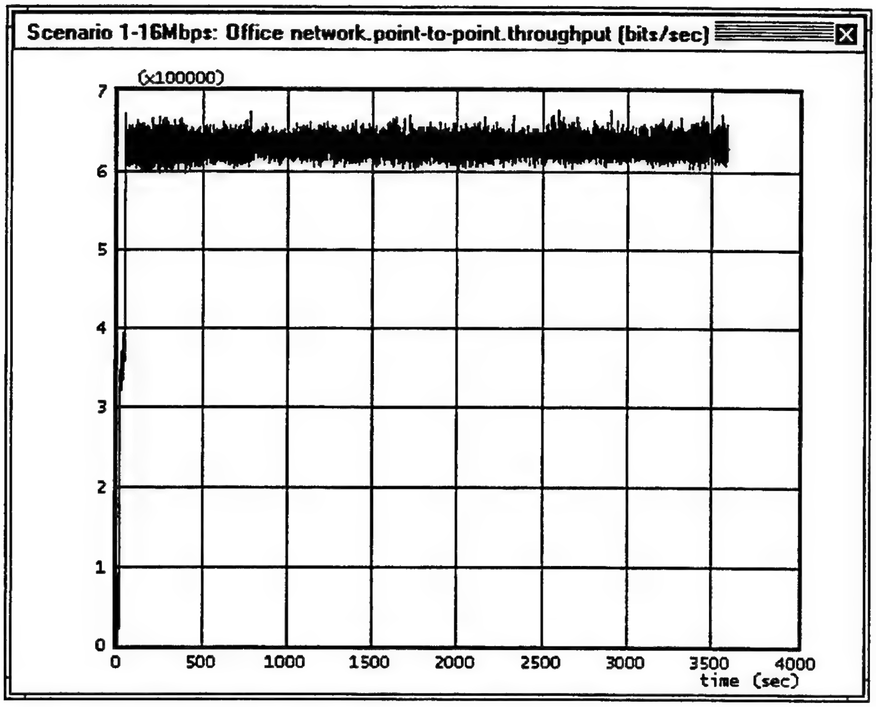

Figure 16. Data Throughput Simulation Results for Scenario 1.

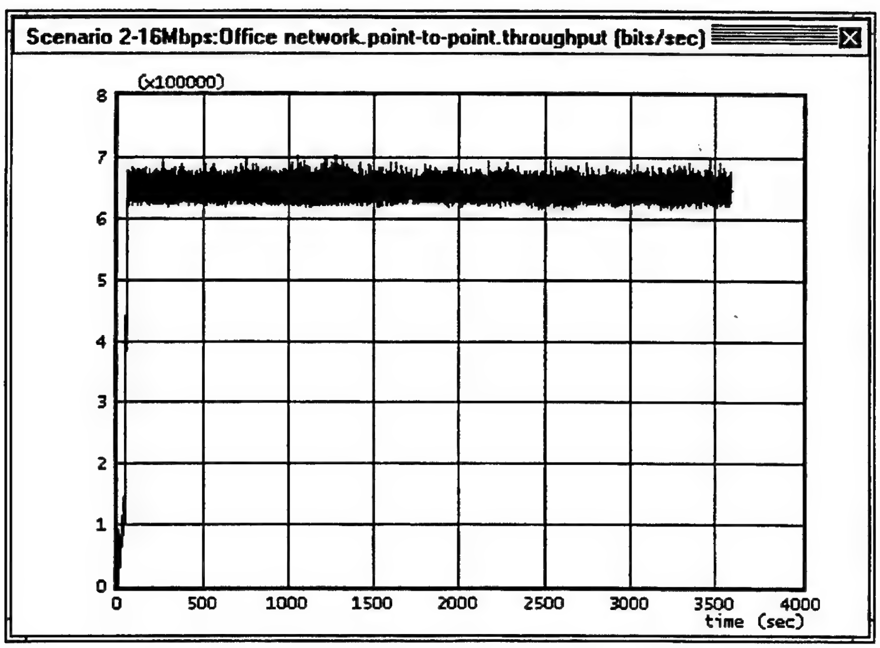

Figure 17. Data Throughput Simulation Results for Scenario 2.

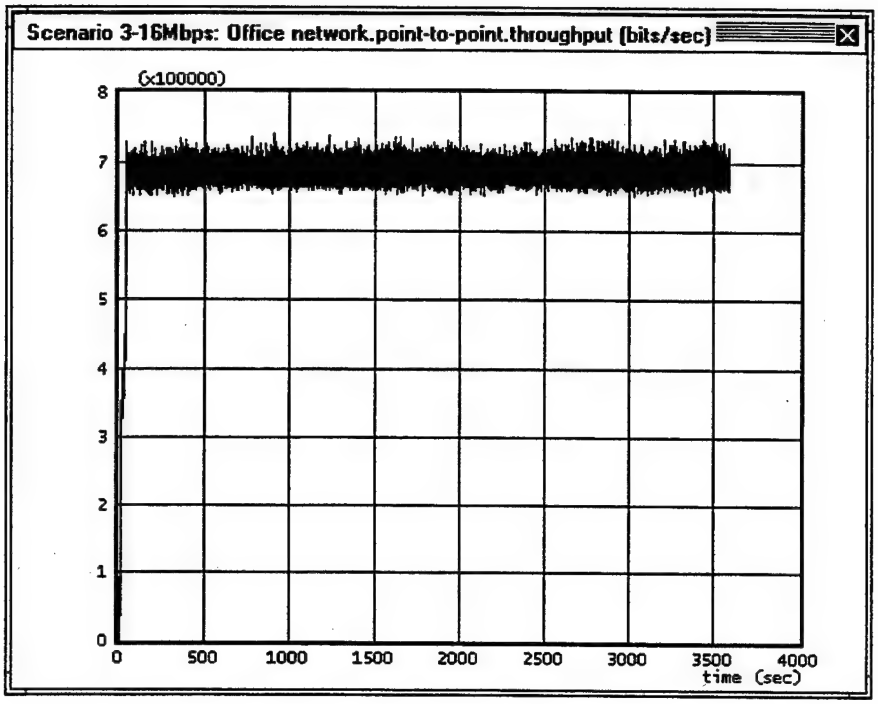

Figure 18. Data Throughput Simulation Results for Scenario 3.

Scenarios 4 and 5 add the Operational Message Traffic to Scenarios 2 and 3. The maximum throughput is measured to be 720.7Kbps and 750.4Kbps during Scenarios 4 and 5, respectively. Figure 19 and Figure 20 contain the simulated network throughput results of these scenarios. As shown in the figures, the network data throughput never exceeds 16Mbps. Table 20 identifies the maximum throughputs measured for Scenarios 4 and 5.

	Scenario	
	(4)	**(5)**
Maximum Link Capacity	**Maximum Observed Throughput (bits/sec)**	
16 Mbits/sec	720.7K	750.4K
32 Mbits/sec	720.7K	750.4K
48 Mbits/sec	720.7K	750.4K
64 Mbits/sec	720.7K	750.4K

Table 20. Summary of Data Throughput Simulation Results for Scenarios 4 and 5.

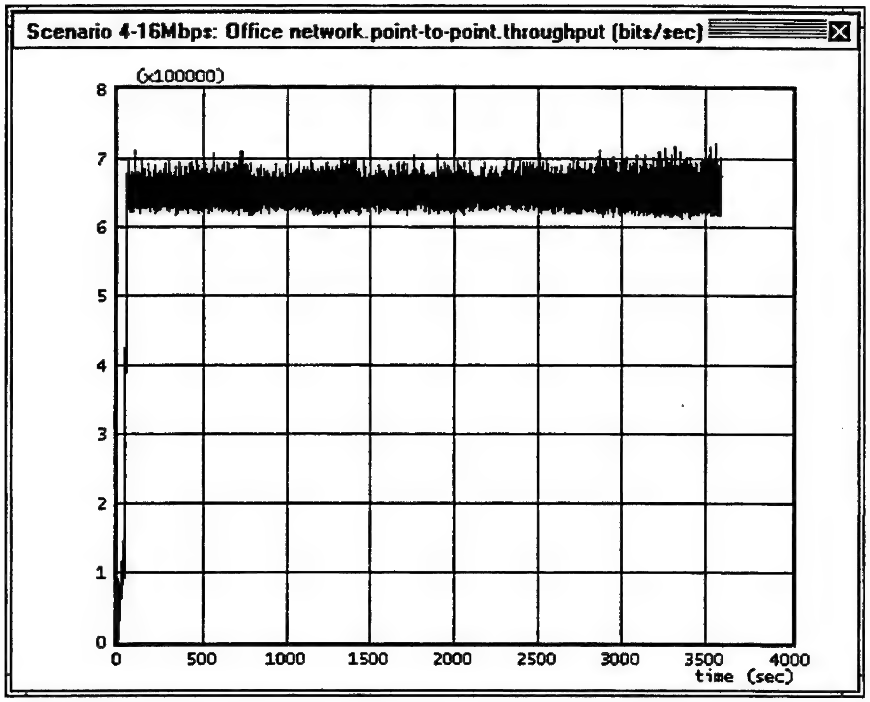

Figure 19. Data Throughput Simulation Results for Scenario 4.

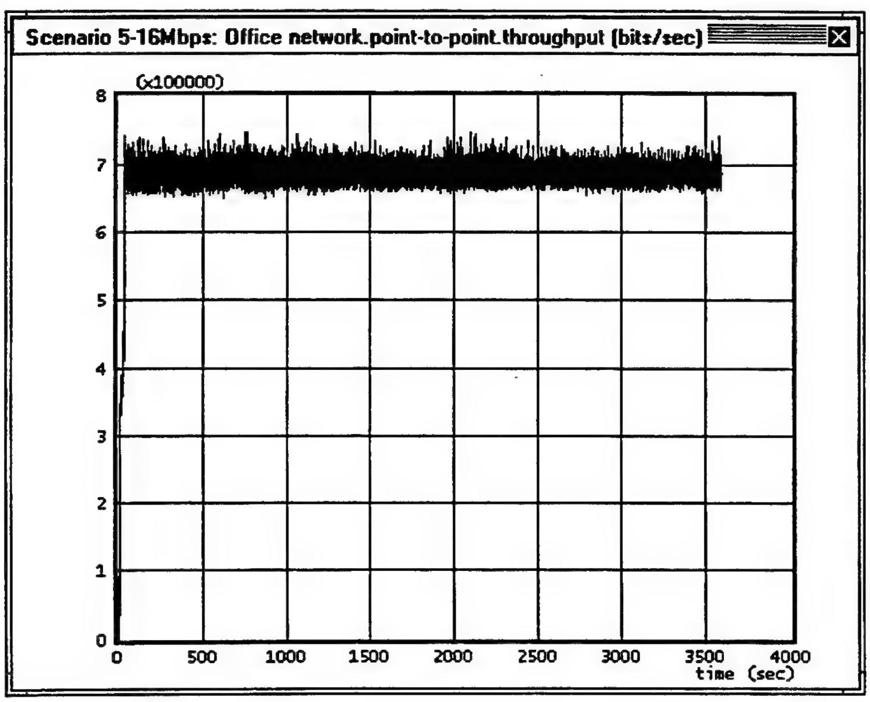

Figure 20. Data Throughput Simulation Results for Scenario 5.

Scenarios 6 and 7 simulate the addition of the 500 Kbyte map database messages to low and high Utility Bus loads, respectively. The simulation results for Scenarios 6 and 7 are shown in Figure 21 and Figure 22, respectively. These figures show a close-up view of the simulations to illustrate the throughput peaks realized when the map database messages are transmitted by the CCS. Examination of Figure 21 reveals that the maximum data throughput simulated during Scenario 6 is measured to be 17.03 Mbps. Figure 22 shows the maximum throughput during simulation of Scenario 7 to be 19.12 Mbps. Therefore, a 32 Mbps network bandwidth is sufficient for the message traffic under both scenarios. Additionally, a closer look (not shown) at these graphs reveals that the transmission time for each map database message ranges between 0.5 and 0.75 seconds.

The maximum throughput measured for each of these scenarios is presented in Table 21. Notice that the data throughput measurements for the 16Mbps network are greater than the network bandwidth. This anomaly is due to the process by which OPNET simulates and collects the network throughput statistic. However, the data throughput observed in the simulation results are considered to be credible and to correctly reflect the traffic load placed on the network.

	Scenario	
	(6)	**(7)**
Maximum Link Capacity	**Maximum Observed Throughput (bits/sec)**	
16 Mbits/sec	17.03M	17.08M
32 Mbits/sec	17.01M	19.12M
48 Mbits/sec	17.03M	17.08M
64 Mbits/sec	17.03M	17.08M

Table 21. Summary of Data Throughput Simulation Results for Scenarios 6 and 7.

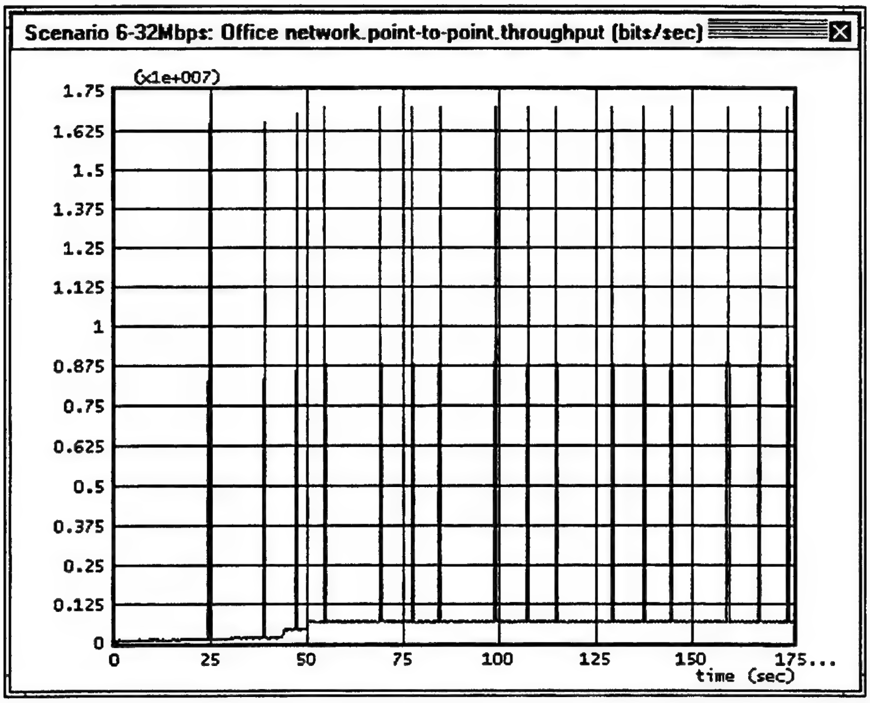

Figure 21. Data Throughput Simulation Results for Scenario 6.

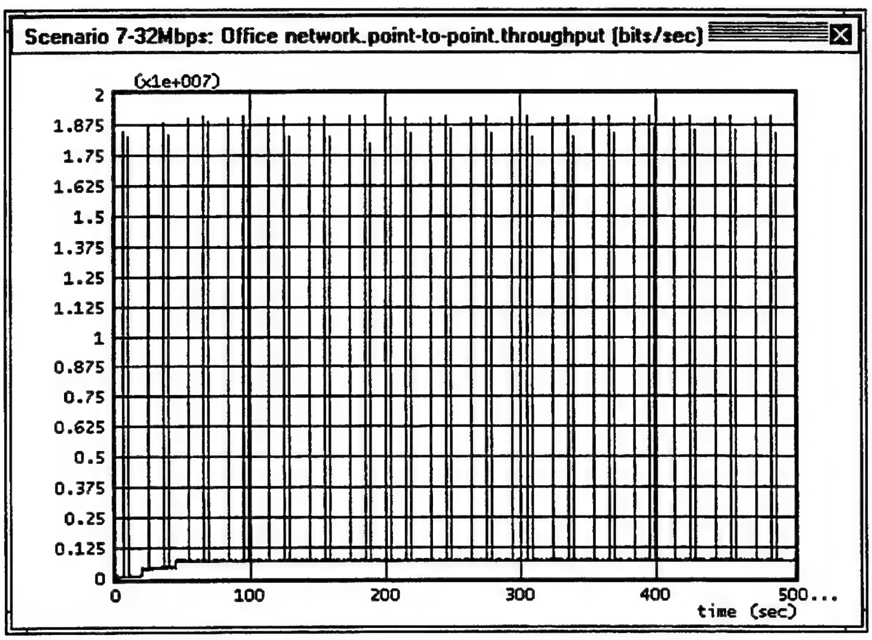

Figure 22. Data Throughput Simulation Results for Scenario 7.

As described in Chapter IV, Scenarios 8 and 9 are similar to Scenarios 6 and 7 except that the map database messages are modeled to be 1.0 Mbytes in size. A detailed view of the data throughput simulation results of Scenario 8 is displayed in Figure 23 to illustrate the data throughput peaks experienced when the map database messages are transmitted. The maximum throughput achieved during the simulations is measured to be 37.46 Mbps. Analysis of the simulation results shows that the transmission time of each map database message ranges between 0.5 and 0.75 seconds. The maximum throughput achieved during the simulations of Scenario 9 is measured to be 37.5 Mbps. The simulation results are displayed in Figure 24 which shows the entire 3600 second simulation to illustrate the consistency of the throughput peaks. The simulation results confirm that a 48 Mbps network bandwidth is sufficient for the message traffic of Scenarios 8 and 9. The maximum data throughput measurements for Scenarios 8 and 9 are presented in Table 22.

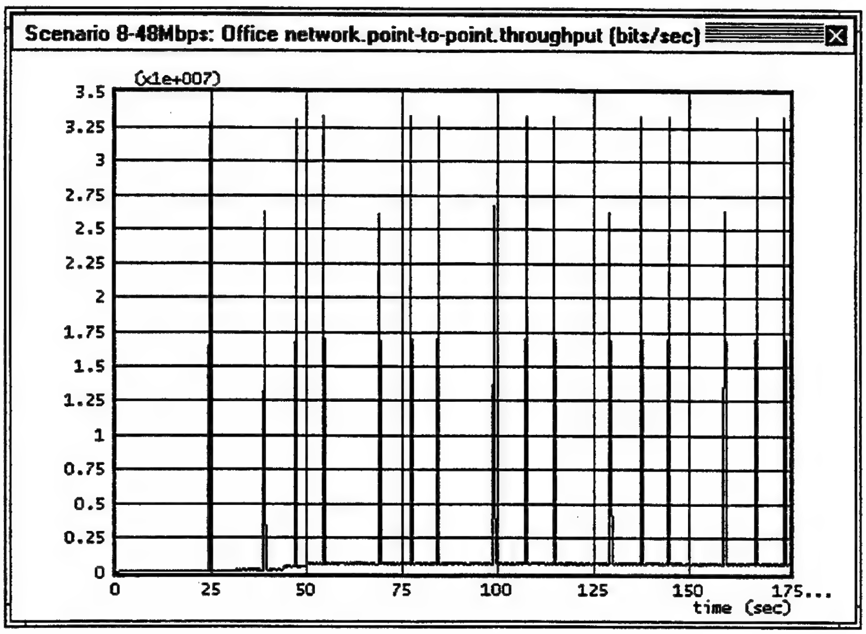

Figure 23. Data Throughput Simulation Results for Scenario 8.

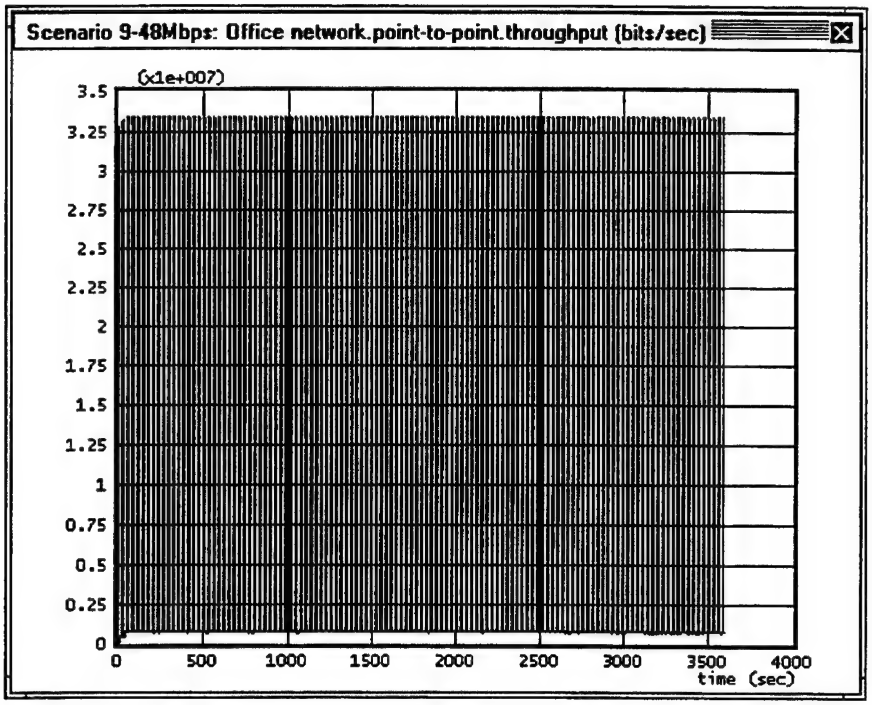

Figure 24. Data Throughput Simulation Results for Scenario 9.

	Scenario	
	(8)	(9)
Maximum Link Capacity	Maximum Observed Throughput (bits/sec)	
16 Mbits/sec	33.38M	33.43M
32 Mbits/sec	37.46M	37.50M
48 Mbits/sec	33.38M	33.43M
64 Mbits/sec	33.38M	33.43M

Table 22. Summary of Data Throughput Simulation Results for Scenarios 8 and 9.

Scenarios 10 and 11 simulate the addition of the Battlesight and Boresight Messages to Scenarios 7 and 9. It is expected that the additional message traffic load would be minimal. Therefore, the results are anticipated to be very similar to those of Scenarios 7 and 9. This is confirmed by examining the simulation results, shown in Figure 25 and Figure 26. The network throughput is slightly higher than that simulated during Scenarios 7 and 9. The simulation results show that a 32 Mbps bandwidth is sufficient for Scenario 10 message

traffic, and a 48Mbps bandwidth is sufficient for the message traffic of Scenario 11. Table 23 presents a summary of the simulation results for Scenarios 10 and 11.

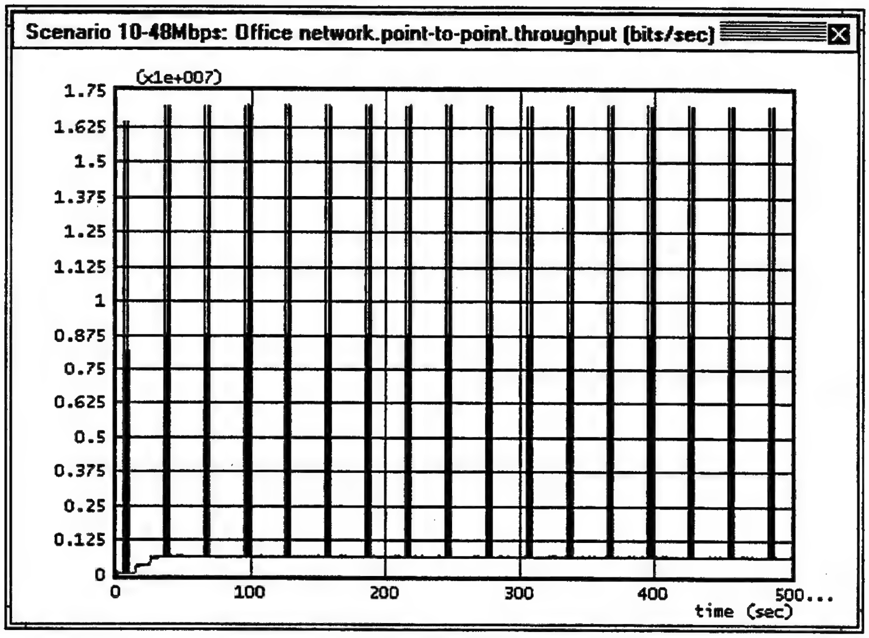

Figure 25. Data Throughput Simulation Results for Scenario 10.

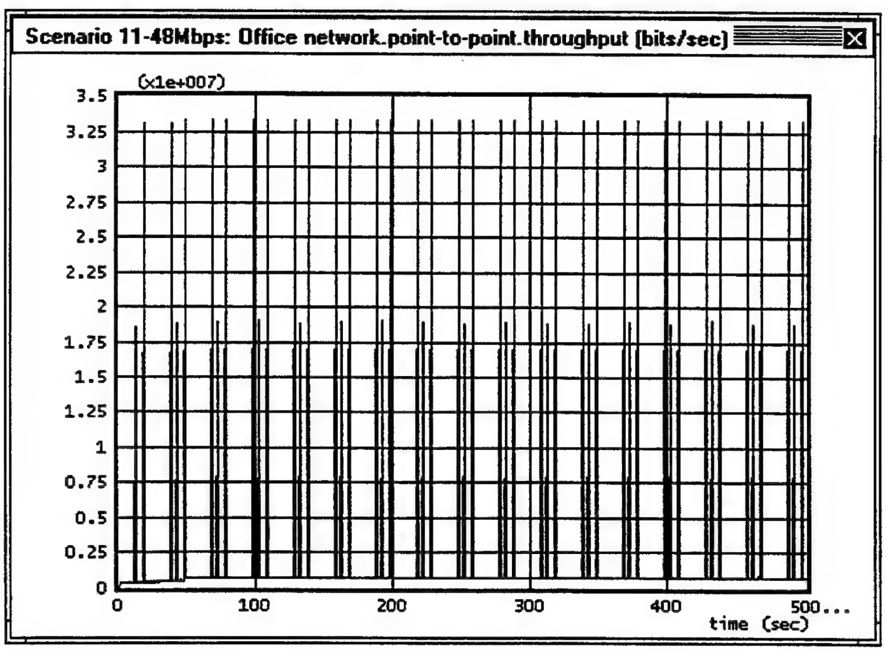

Figure 26. Data Throughput Simulation Results for Scenario 11.

	Scenario	
	(10)	(11)
Maximum Link Capacity	**Maximum Observed Throughput (bits/sec)**	
16 Mbits/sec	17.10M	33.42M
32 Mbits/sec	19.13M	37.52M
48 Mbits/sec	17.10M	33.42M
64 Mbits/sec	17.10M	33.42M

Table 23. Summary of Data Throughput Simulation Results for Scenarios 10 and 11.

2. High Speed Data Bus Queue Sizes

The maximum queue sizes during each of the scenario simulations are observed to determine the AAAV-P hardware requirements in order to prevent overflow. As explained previously, the TEU, HEU, and WSEU hardware differs from the CCS hardware. Therefore, the queue size measurements are reported as the maximum of the TEU, HEU, and WSEU simulation results, while separate queue size measurements are collected and reported for the CCS.

Most simulation results are unremarkable with respect to the queue sizes. The simulation results can be summarized by describing a few scenario results.

Figure 27 displays the simulation results of Scenario 1, specifically the statistics for all four MAC queue sizes. Since the link capacity is much greater than the traffic load imposed on the network, the queues typically hold only one or two messages awaiting service. Analysis of Figure 27 shows that the queue sizes are small (approximately two packets in length) and relatively constant, as expected.

Figure 28 displays the simulation results of Scenario 6 for the TEU, WSEU, and HEU MAC queue sizes. The peaks correspond to the transmission of map database messages from the CCS.

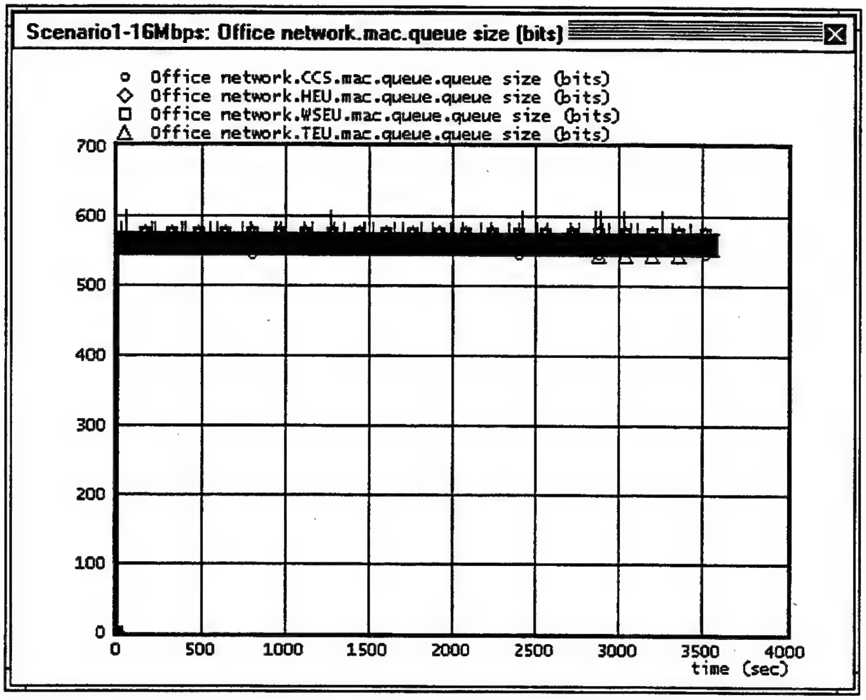

Figure 27. Queue Size Simulation Results of Scenario 1.

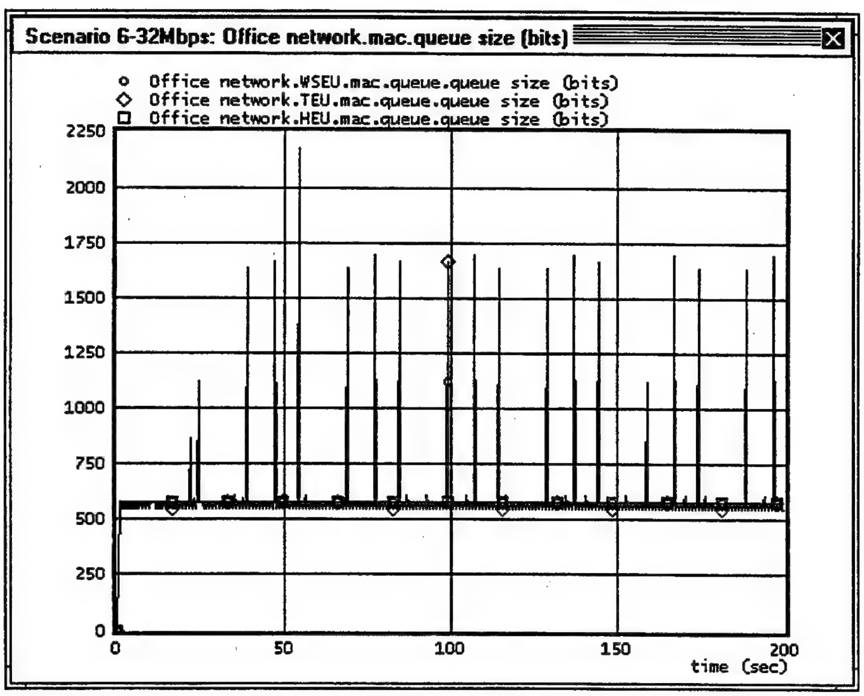

Figure 28. TEU, WSEU, and HEU Queue Size Simulation Results of Scenario 6.

Figure 29 shows the simulation results of Scenario 6 for the CCS MAC queue size. These results display a small, constant queue size except when the CCS periodically transmits the map database messages.

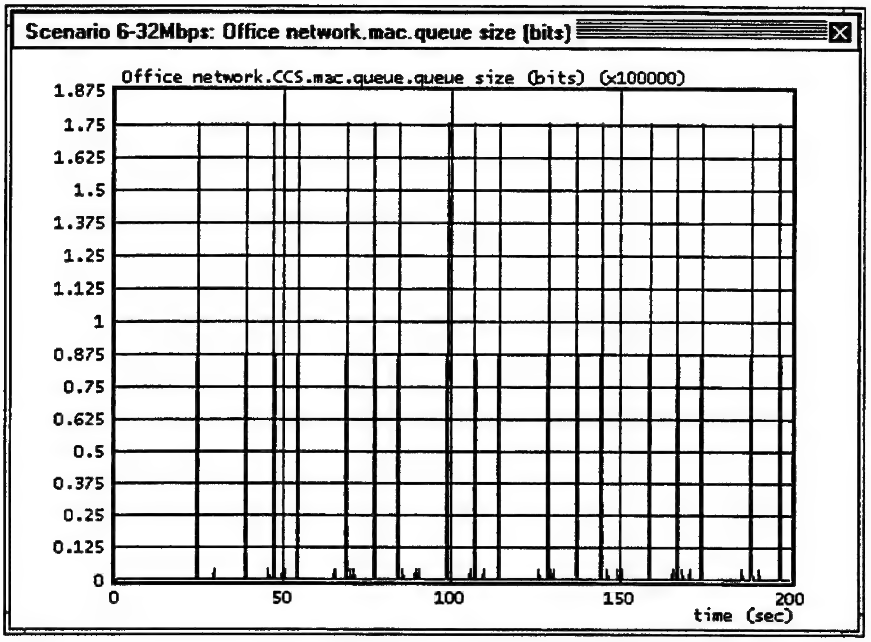

Figure 29. CCS Queue Size Simulation Results of Scenario 6.

Through analysis of the simulation results of the eleven scenarios, the maximum queue length experienced by the TEU, HEU, or WSEU is measured to be 2816 bits. The maximum queue length experienced by the CCS is measured to be 175.744 Kbits. All of the queue size measurements are displayed in Table 24 and Table 25 at the end of this chapter.

3. PITCH_ANGLE_FDDI and ROLL_ANGLE_FDDI Message Arrival Rates

The messages transmitted at the highest frequency are the PITCH_ANGLE_FDDI and the ROLL_ANGLE_FDDI messages. These messages are transmitted by the HEU and sent to the WSEU at a frequency of 720,000 times per hour, or once every 5msec. It is

important that the frequency at which the messages are received is at a periodic and constant rate with minimal delay jitter.

A conceptual example follows. The PITCH_ANGLE_FDDI message is transmitted at a constant rate of once every 5 msec. The first message is transmitted immediately, experiencing various processing (i.e., TCP or IP) delays and a FDDI transmission delay. When the next message is generated 5 msec after the first message, it is held in the MAC queue while the node waits for access to the network. This results in a MAC queue delay in addition to the processing and FDDI transmission delays. The following message is then transmitted 5 msec after the previous message and experiences only processing and FDDI transmission delays. This varying pattern may continue throughout the simulation but is most pronounced when the bus is heavily loaded (e.g., during map database message transmissions). Figure 30 illustrates the fluctuations that may occur with respect to the period between arriving messages. As shown, it is possible that the transmission of the next message will never occur before arrival of the previously transmitted message. However, it is also possible for the total delay to be large enough such that a new message transmission occurs before the previous message is received at the destination.

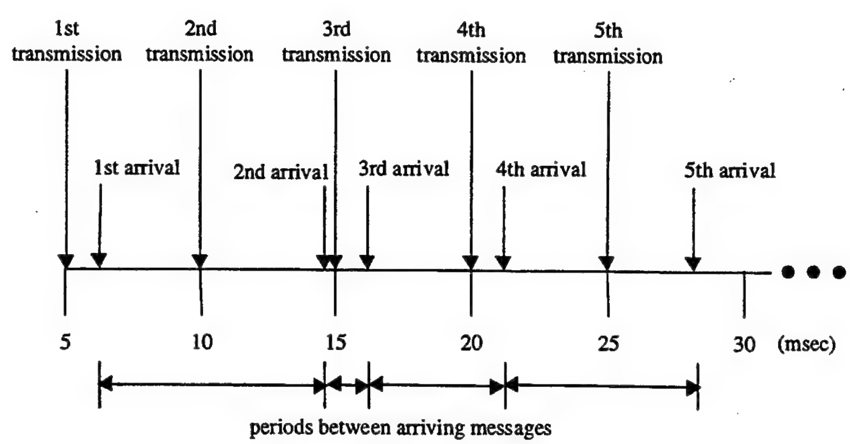

Figure 30. Illustration of Fluctuations in Message Arrival Rates.

Measuring PITCH_ANGLE_FDDI and ROLL_ANGLE_FDDI message arrivals involved modifying the user-defined message generator and message receiver process models. The *heu_msg_gen* process model was modified to include message information in the data field of the AAAV-FDDI_pk packet. This message information includes the message name. The *aaav_msg_rcvr* process model was modified to extract the message name from the data field and calculate the time between arrivals of the PITCH_ANGLE_FDDI and the ROLL_ANGLE_FDDI messages. The modified code is contained in Appendix H.

As shown in Figure 31, the arrival rates of the two messages are affected when the map database messages are transmitted.

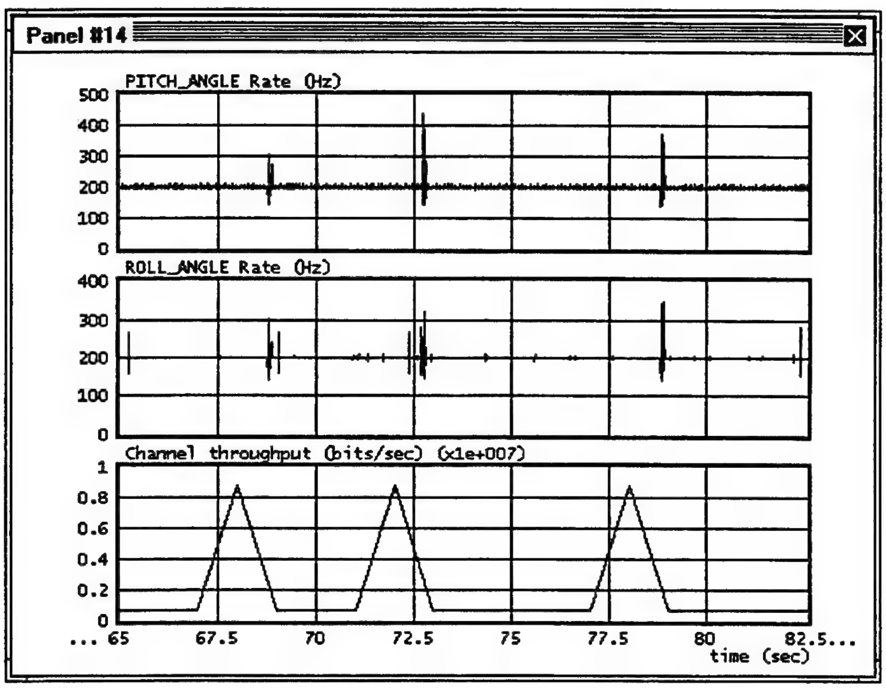

Figure 31. Simulated PITCH and ROLL_ANGLE_FDDI Message Arrival Rates.

The rate at which the PITCH_ANGLE_FDDI and ROLL_ANGLE_FDDI messages are transmitted is 200 Hz. The simulation results show that when the map database

messages are not being transmitted, the arrival rate shows only small variations and typically ranges between 190 Hz and 210 Hz. Analysis of Figure 31 shows the correlation between the variability in arrival rates and the observed throughput peaks. Analysis of the simulation results shows that the arrival rate ranges between 139 Hz and 442 Hz when map database messages are transmitted. Therefore, the simulated time between arriving messages ranges from 2.26 msec (1/442Hz) to 7.19 msec (1/139Hz). When a map database message is transmitted, the PITCH_ANGLE_FDDI and ROLL_ANGLE_FDDI messages are delayed in the MAC queue. The arrival rates then fluctuate as the messages are held in the queue while the channel is busy and then transmitted as soon as the channel bandwidth is available. These results are shown in Figure 32, which is a close up view of the top two graphs shown in Figure 31.

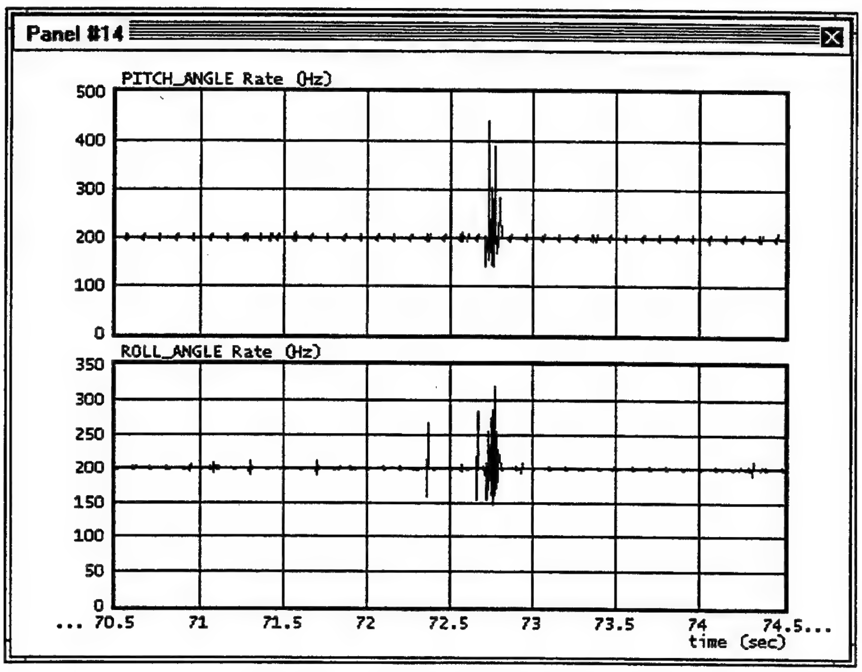

Figure 32. Close Up View of PITCH and ROLL_ANGLE_FDDI Message Arrival Rates.

These results were collected while the network was simulated with a 48 Mbps channel. The AAAV Project Office should consider whether this 7.19 msec period between

arriving messages is acceptable during operation. It is important to note that the 7.19 msec period between arriving messages is not constant. This period only occurs during the time a map database message is transmitted. Additionally, the 7.19 msec period is followed by arrival rates above 200 Hz (e.g., peaks of 442 Hz in one instance), and only as a series of transient spikes. Additionally, it is important to note the uncertainty regarding the relationship between the arriving message and the next transmission. The simulations do not confirm that the arrival of a message occurs before the transmission of the next message. This can be determined through additional modeling and simulation efforts if the AAAV Project Office is concerned regarding the demonstrated delay jitter.

If a maximum delay of 7.19 msec is not acceptable, a possible alternative is to smooth the transmission of the map database messages to make transmissions less bursty. This alternative and its potential benefits are further explained in the next chapter.

C. SUMMARY OF SIMULATION STATISTICS

Table 24 and Table 25 summarize the statistics collected during the simulations described above. As shown in the table, the network throughput consistently increases as additional message traffic is imposed on the network. The results also show that the simulated link throughput sometimes exceeds the available link bandwidth for Scenarios 6 through 11. MIL3 stated that this is a result of the simulation and statistics collection process within OPNET. However, the throughput results are considered credible in identifying the maximum required bandwidth necessary to support (i.e., avoid packet loss and minimize message delay jitter) the associated message traffic on the network. The data throughput results of Scenarios 6 through 11 consistently differ when the scenario is

simulated on a 32 Mbps channel. This is clearly an anomaly. However, the difference is minimal and should not be considered significant.

The results tables show that the simulated queue sizes increase as the network message traffic increases. However, it is expected that the queue sizes would decrease as the network bandwidth increases for the same traffic load. In most instances, the queue sizes remain constant. This anomaly may need to be investigated further to eliminate uncertainties associated with queue size requirements.

	Scenario				
	(1)	(2)	(3)	(4)	(5)
Maximum Link Capacity	**Maximum Observed Throughput (bits/sec)**				
16 Mbits/sec	679.3K	704.6K	740.5K	720.7K	750.4K
32 Mbits/sec	679.3K	704.6K	740.5K	720.7K	750.4K
48 Mbits/sec	679.3K	704.6K	740.5K	720.7K	750.4K
64 Mbits/sec	679.3K	704.6K	740.5K	720.7K	750.4K
	Maximum MAC Queue Length (bits)				
	TEU, WSEU, HEU				
16 Mbits/sec	608	864	864	864	864
32 Mbits/sec	608	864	864	864	864
48 Mbits/sec	608	864	864	864	864
64 Mbits/sec	608	864	864	864	864
	CCS				
16 Mbits/sec	576	608	576	4544	4544
32 Mbits/sec	576	608	576	4544	4544
48 Mbits/sec	576	608	576	4544	4544
64 Mbits/sec	576	608	576	4544	4544

Table 24. Summary of Simulation Results of Scenarios 1, 2, 3, 4, and 5.

	Scenario					
	(6)	(7)	(8)	(9)	(10)	(11)
Maximum Link Capacity	**Maximum Observed Throughput (bits/sec)**					
16 Mbits/sec	17.03M	17.08M	33.38M	33.43M	17.01M	33.42M
32 Mbits/sec	17.01M	19.12M	37.46M	37.50M	19.13M	37.52M
48 Mbits/sec	17.03M	17.08M	33.38M	33.43M	17.10M	33.42M
64 Mbits/sec	17.03M	17.08M	33.38M	33.43M	17.10M	33.42M
	Maximum MAC Queue Length (bits)					
	TEU, WSEU, HEU					
16 Mbits/sec	2816	2240	2816	2272	2240	2784
32 Mbits/sec	2816	2240	2816	2272	2240	2784
48 Mbits/sec	2816	2240	2816	2272	2240	2784
64 Mbits/sec	2816	2240	2816	2272	2240	1664
	CCS					
16 Mbits/sec	175.7K	175.7K	175.7K	175.8K	175.7K	175.7K
32 Mbits/sec	175.7K	175.7K	175.7K	175.8K	175.7K	175.7K
48 Mbits/sec	175.7K	175.7K	175.7K	175.8K	175.7K	175.7K
64 Mbits/sec	175.7K	175.7K	175.7K	175.8K	175.7K	175.7K

Table 25. Summary of Simulation Results of Scenarios 6, 7, 8, 9, 10, and 11.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

The purpose of this research was to analyze the performance capability of the AAAV-P Vetronics System High Speed Data Bus. This was accomplished using OPNET, which provided valuable insight into the performance capability of the network. The simulation results showed maximum throughputs of 19.13 Mbps and 37.52 Mbps, depending on the map database message size simulated. If the map database messages are 500 Kbytes in size, the results indicate that a 32 Mbps channel would be sufficient for the message traffic identified for this research. If the map database messages are 1.0 Mbytes in size, the simulation results indicate that a 48 Mbps channel would be sufficient for the message traffic identified for this research. If it is later determined that the size of the map database messages exceeds 1.0 Mbytes, it is recommended that additional simulations be performed. The results presented in this thesis should be weighed against the associated costs of the 32, 48, and 64 Mbps channels.

The simulation results also provide information regarding hardware requirements, specifically MAC queue requirements. The results indicate that the TEU, WSEU, and HEU MAC queue requirements are much lower than the CCS MAC queue requirements. During the simulations, the TEU, WSEU, and HEU MAC queues reached a maximum length of 2816 bits, while the CCS MAC queue reached a maximum length of 175.8 Kbits during the transmission of the 1.0 Mbyte map database messages. As noted previously, the simulation results identify the same queue sizes, regardless of whether the link is saturated or unsaturated. This anomaly needs to be investigated further to eliminate uncertainties associated with queue size requirements.

The arrival rates of specific messages were monitored to determine the periodicity and consistency of the arriving messages. The PITCH_ANGLE_FDDI and ROLL_ANGLE_FDDI message arrival times were collected at the WSEU network node to determine the arrival rate of the messages. These messages are transmitted by the WSEU to the HEU every 5.0 msec. Ideally, the arrival rate is deterministic. The simulation results indicate that the arrival rates are most affected when the map database messages are transmitted. The maximum period between arriving messages was 7.19 msec which corresponds to a 139 Hz arrival rate. This minimal arrival rate was realized during the transmission of a large map database message. Otherwise, the arrival rates of the messages were relatively constant and ranged between 190 Hz and 210 Hz. It is recommended that the AAAV Project Office consider the maximum delay jitter of 7.19 msec and determine if further investigation is necessary.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

Based on the experiences of this effort, OPNET's modeling software appears to be a useful tool for network modeling and simulation. It is a difficult task to understand and utilize OPNET's modeling and analysis tools. However, once their basic usage is understood, OPNET is an essential tool for any network modeling effort.

Additional modeling efforts are recommended to further analyze the performance of the Vetronics System communications network and to explore alternative network designs to enhance the network performance. Additionally, several network model enhancements are recommended to improve the capabilities of the existing network model. These are described below. Follow on efforts should also be conducted to explore the throughput and queue size anomalies described previously in this thesis.

1. Additional Modeling Efforts

This modeling effort was designed to be a starting point, offering a foundation upon which a more detailed simulation of the Vetronics Communications System might be built. The design approach of this effort emphasized a building block approach upon which additional network models can be incorporated. Future work might include developing a model and analyzing the performance of the Versa Module European (VME) Bus within each of the TEU and HEU nodes. The VME Bus facilitates the communications between the High Speed Data Bus and the Utility and CAN buses. Additionally, the VME Bus within the CCS as well as the CSCI channels connecting the MMU and the DACT to the CCS can be modeled and the bandwidth requirements identified. Future work may also include a simulation of the Utility and CAN Buses. Integrating these models with the High Speed Data Bus model may provide additional insight into their impact on the High Speed Data Bus network throughput. To accomplish the current research efforts, assumptions were made with respect to the High Speed Data Bus messages resulting from occurrences on the Utility and CAN Buses. If these assumptions are correct, the insight gained from modeling the Utility and CAN Buses would be minimal and is not recommended.

2. Modifications to Existing High Speed Data Bus Network Model

Several design choices were made when developing the model for the High Speed Data Bus. As development progressed and simulations were executed, it became apparent that specific enhancements to the model would facilitate future simulation efforts. A specific recommended change is to enhance the *msg_rcvr* module to include message processing capabilities. Currently, the process model receives and discards the messages. Enhancements would involve reading the message fields and processing the information as

necessary, such as enabling a destination node to respond to a specific message with another message to the source node. The modified code in Appendix H provides an example of enhanced model capabilities.

Future work may also include investigating methods to reduce the bandwidth requirements. A possible solution is to smooth the transmission of each map database message over a specified time frame. Implemented in the application software, this should reduce the transmission peaks illustrated throughout the simulation results of this research. The distribution of the transmission of the map database messages should also reduce the required CCS MAC queue size as well as decrease the maximum delay experienced by the PITCH_ANGLE_FDDI and ROLL_ANGLE_FDDI messages. A major advantage of this scheme is that changes to the actual software would likely be localized to the database transmission software. This is in contrast to implementing FDDI priorities (as described below) which would likely be more pervasive.

Using a FDDI priority scheme would allow important messages to be sent prior to large map database messages. According to Jain [8], four conditions must exist concurrently for priorities to be effective:

1. High load.
2. Short packets.
3. Nonexhaustive service.
4. Small number of stations on the network.

A high load exists periodically when the CCS is generating map database messages for transmission. The use of short packets leads to many frames competing for the network. A nonexhaustive service network is one in which the amount of time each station can transmit

is fixed (i.e., a THT is implemented). In an exhaustive service network, each station is permitted to send all frames in the queue before the next queue is allowed to transmit its frames. Finally, if the number of stations is small, the total number of frames waiting is divided among only a few stations and the average queue length per station is large. Priorities can be assigned to frames as well as to stations. [8] However, this method may be more pervasive than distributing the transmission of each map database message over a specified time frame.

According to Jain [8], the TTRT and frame size can be adjusted to optimize the network's performance. Increasing the TTRT improves efficiency but also increases the maximum access delay. Jain states that a TTRT equal to 8 msec is recommended to maximize performance in all ranges of configurations. [8] This theory was not explored during this research effort. However, future modeling efforts may include extra simulations in order to investigate this theory.

APPENDIX A. ACRONYMS AND ABBREVIATIONS

This Appendix provides a list of all acronyms and abbreviations used throughout this thesis.

AAAV	Advanced Amphibious Assault Vehicle
AAAV-C	Advanced Amphibious Assault Vehicle Command and Control Variant
AAAV-P	Advanced Amphibious Assault Vehicle Personnel Variant
ACK	Acknowledgement
AFES	Automatic Fire Extinguishing System
AMB	Analog Monitor Board
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
bps	Bits Per Second
CAN	Controller Area Network
CB	CAN Bus
CCS	Command and Control Server
CD	Controls and Displays
CDP	Control Display Panel
CSCI	Computer Software Configuration Item
DA	Destination Address
DACT	Data Automated Communications Terminal
DRPM	Direct Reporting Program Manager
EMI	Electromagnetic Interference
EPLRS	Enhanced Positioning Location Reporting System
FC	Frame Control
FDDI	Fiber Distributed Data Interface
FSM	Finite State Machine
GDF	General Data File
GPP	General Purpose Processor
GPS	Global Positioning System
HEU	Hull Electronics Unit

HPDU	Hull Power Distribution Unit
HRACM	Hull Remote Acquisition Control Module
ICI	Interface Control Information
IP	Internet Protocol
JMCIS	Joint Maritime Command Information System
Kbps	Kilobits per second
lsb	Least Significant Bit
MAC	Medium Access Control
Mbps	Megabits Per Second
Mbytes	Megabytes
MMU	Mass Memory Unit
MPA	Mobility/Power Management/Auxiliary
msb	Most Significant Bit
Msec	Milliseconds
NAV	Navigation
NBC	Nuclear, Biological, Chemical
NRZI	Non-Return to Zero Inverted
OPNET	Optimized Network Engineering Tool
pps	Packets Per Second
PUIT	Packet Update Identification Tag
RACM	Remote Acquisition Control Module
RDD	Requirements Definition Document
SA	Situational Awareness
SA	Source Address
SINCGARS	Single Channel Ground Airborne Radios System
SK	Simulation Kernel
TCP	Transport Control Protocol
TEU	Turret Electronics Unit
TPAL	Transport Adaptation Layer
TRACM	Turret Remote Acquisition Control Module
UB	Utility Bus

UDP	User Data Protocol
USMC	United States Marine Corps
VME	Versa Module European
WSEU	Weapon Station Electronics Unit

APPENDIX B. HIGH SPEED DATA BUS FRAME FORMATS

This appendix provides a detailed description of the frame formats used in the implementation of TCP/IP over a FDDI MAC. The formats identified are the token frame format, FDDI header/trailer format, IP header format, TCP header format, and the AAAV specific data format. Each of these formats is described to the byte level.

A. 4B/5B NRZI ENCODING ALGORITHM

The AAAV-P High Speed Data Bus communicates using symbols and a 4B/5B Non-Return to Zero Inverted (NRZI) encoding algorithm. Encoding is executed four bits at a time, and each four bits of data are encoded into a five-bit symbol sequence. Symbols represent 16 data symbols and 8 control symbols. These are listed in Table 26. The symbols are assigned such that a station will never receive four consecutive zeros. [23]

Data Symbols	**bit stream**
0 (binary 0000)	11110
1 (binary 0001)	01001
2 (binary 0010)	10100
3 (binary 0011)	10101
4 (binary 0100)	01010
5 (binary 0101)	01011
6 (binary 0110)	01110
7 (binary 0111)	01111
8 (binary 1000)	10010
9 (binary 1001)	10011
A (binary 1010)	10110
B (binary 1011)	10111
C (binary 1100)	11010
D (binary 1101)	11011
E (binary 1110)	11100
F (binary 1111)	11101
Control Symbols	
Q	00000
H	00100
I	11111

J	11000
K	10001
T	01101
R	00111
S	11001

Table 26. FDDI Symbol Encoding Scheme. [24].

B. FDDI TOKEN FORMAT

Starting Delimiter (1 Byte)	Frame Control (1 Byte)	Ending Delimiter (1 Byte)

The token is preceded by a "Preamble" that synchronizes the frame with each station's clock [23]. The Preamble is two or more bytes in length. Table 27 presents the breakdown of the FDDI token frame format.

Byte	Bit Order	Field ID	Values (HEX)	Description
1	msb	Starting Delimiter	JK	• Indicates the start of a token frame
2	msb	Frame Control (FC)	80 or C0	• 80 indicates a non-restricted token • C0 indicates a restricted token
3	lsb	Ending Delimiter	TT	• Indicates the end of a token frame

Table 27. Token Frame Format. [6]

C. FDDI HEADER/TRAILER FORMAT

FDDI Header (14 Bytes)	IP Header (20 Bytes)	TCP Header (20 Bytes)	AAAV Specific Data (up to 4440 Bytes)	FDDI Trailer (6 Bytes)

Transmission of a FDDI frame is preceded by a Preamble. Table 28 and Table 29 present the breakdown of the FDDI header and trailer formats respectively.

Byte	Bit Order	Field ID	Values (HEX)	Description
1	msb	Starting Delimiter (SD)	JK	• Indicates the start of the frame

Byte	Bit Order	Field ID	Values (HEX)	Description
2	msb	Frame Control (FC)	See App. A	• Indicates the type of data in the AAAV Specific Data field
3	lsb	Destination Address (DA)	00 or FF	• 00 (lsb = 00) is part of the IEEE assigned address for Vista Controls • FF is part of the "broadcast" address[10]
4	lsb	Destination Address (DA)	02 or FF	• 02 (lsb = 40) is part of the IEEE assigned address for Vista Controls • FF is part of the "broadcast" address
5	lsb	Destination Address (DA)	03 or FF	• 03 (lsb = C0) is part of the IEEE assigned address for Vista Controls • FF is part of the "broadcast" address
6	lsb	Destination Address (DA)	"aa" or FF	• "aa" is part of the address assigned each FDDI mezzanine[11] • FF is part of the "broadcast" address
7	lsb	Destination Address (DA)	"bb" or FF	• "bb" is part of the address assigned each FDDI mezzanine • FF is part of the "broadcast" address
8	lsb	Destination Address (DA)	"cc" or FF	• "cc" is part of the address assigned each FDDI mezzanine • FF is part of the "broadcast" address
9	lsb	Source Address (SA)	00	• 00 (lsb = 00) is part of the IEEE assigned address for Vista Controls
10	lsb	Source Address (SA)	02	• 02 (lsb = 40) is part of the IEEE assigned address for Vista Controls

[10] FDDI supports individual addressing, group addressing, and broadcast (all stations to receive) addressing. The AAAV High Speed Data Bus will use individual and broadcast addressing only.

[11] Each FDDI mezzanine is assigned (by the board manufacturer) a unique physical address (e.g., a HEU in one AAAV will have a different physical address than a HEU in another AAAV).

Byte	Bit Order	Field ID	Values (HEX)	Description
11	lsb	Source Address (SA)	03	• 03 (lsb = C0) is part of the IEEE assigned address for Vista Controls
12	lsb	Source Address (SA)	"aa"	• "aa" is part of the address assigned each FDDI mezzanine
13	lsb	Source Address (SA)	"bb"	• "bb" is part of the address assigned each FDDI mezzanine
14	lsb	Source Address (SA)	"cc"	• "cc" is part of the address assigned each FDDI mezzanine

Table 28. FDDI Header Format. [6].

Byte	Bit Order	Field ID	Values (HEX)	Description
n-5 n-4 n-3 n-2	msb	Frame Check Sequence (FCS)	Variable	• Used by a receiving station to verify that the frame traversed the network without incurring any bit errors.
n-1	msb	Ending Delimiter (ED); Error Indicator	Variable	• The Ending Delimiter (4 Bits) consists of a single 'T' symbol. This 'T' symbol indicates that the frame is complete. • The Error Indicator (4 Bits) is part of the Frame Status field.
n[12]	msb	Acknowledge Indicator Copy Indicator	"aa" or FF	• The Acknowledge and Copy Indicators (4 Bits each) are part of the Frame Status field.

Table 29. FDDI Trailer Format. [6].

D. IP HEADER FORMAT

FDDI Header (14 Bytes)	IP Header (20 Bytes)	TCP Header (20 Bytes)	AAAV Specific Data (up to 4440 Bytes)	FDDI Trailer (6 Bytes)

Table 30 presents the breakdown of the IP header.

[12] "n" represents the total frame length in bytes; the maximum length is 4500 bytes

Byte	Bit Order	Field ID	Values (HEX)	Description
15	msb	IP Version ID; IP Header Length	**Variable**	• The IP Version (4 Bits) shall be shall be provided by the operating system • The IP Header Length (4 Bits) identifies the length of the header, in 32-bit words. This value shall be set to 5H[13]
16	msb	Service Type	Not Used	• Allows the host to tell the subnet what kind of service it wants.
17 18	msb	IP Datagram Total Length	Variable	• Includes the header and data • The maximum length is 4480 bytes (4500 bytes - 20 bytes for the FDDI header/trailer)
19 20	msb	IP Datagram Identification	Variable	• Allows the destination host to determine which datagram a newly arrived fragment belongs to
21 22	msb	Unused (Bit 7) Flags (Bits 6,5); Fragment Offset (Bits 4 - 0) Fragment Offset	Variable	• Bit 6 Flag = Don't Fragment (DF) • Bit 5 Flag = More Fragments (MF) - all fragments, except the last, set this bit. • Fragment Offset tells where in the current datagram the fragment belongs.
23	msb	Time To Live	Variable	• Number of seconds that a datagram can exist
24	msb	Protocol	Variable	• Identifies the transport process to give the datagram to (e.g., TCP (= 6)or UDP (=17))
25 26	msb	Header Checksum	Variable	• CRC checksum on IP Header
27 28 29	msb	Source IP Address	Variable	• Indicates the network and host numbers of the datagram's source
30				

[13] This represents the minimum value of the IP Header (20 Bytes). This value implies the non-use of the option field within the IP Header. These options are not required for the AAAV High Speed Data Bus.

Byte	Bit Order	Field ID	Values (HEX)	Description
31	msb	Destination IP Address	Variable	• Indicates the network and host numbers of the datagram's destination
32				
33				
34				

Table 30. IP Header Format. [6].

E. IP ADDRESSES

Each station on the AAAV High Speed Data Bus has a unique IP address, corresponding to its connection to the High Speed Data Bus. The assigned IP addresses for the AAAV High Speed Data Bus network are identified in Table 31.

Host Name	AAAV High Speed Data Bus IP Address
CCS	205.205.205.50
HEU	205.205.205.30
TEU	205.205.205.20
WSEU	205.205.205.40

Table 31. AAAV High Speed Data Bus IP Address Assignments.

F. TCP HEADER FORMAT

FDDI Header (14 Bytes)	IP Header (20 Bytes)	TCP Header (20 Bytes)	AAAV Specific Data (up to 4440 Bytes)	FDDI Trailer (6 Bytes)

Table 32 presents the TCP header format.

Byte	Bit Order	Field ID	Values (HEX)	Description
35	msb	Source Port ID	Variable	• Identifies the end point within the source station
36				
37	msb	Destination Port ID	Variable	• Identifies the end point within the destination station
38				
39	msb	Sequence Number	Variable	• Specifies the byte's sequence within the segment
40				
41				
42				
43	msb	Acknowledgment Number	Variable	• Specifies the next byte expected
43				

Byte	Bit Order	Field ID	Values (HEX)	Description
44				
45				
46				
47	msb	Header Length (first 4 bits); Reserved (last 4 bits)	Variable	• Identifies how many 32-bit words are contained in the TCP header.
48	msb	Reserved (first 2 bits); TCP Code Bits (last 6 bits): Bit 5 - Urgent Pointer (URG) Bit 4 - Ack Number (ACK) Bit 3 - Pushed Data (PSH) Bit 2 - Reset Connection (RST) Bit 1 - Establish Connection (SYN) Bit 0 - Release Connection (FIN)	Variable	• URG: Indicates that the urgent pointer is in use. • ACK: Indicates that the acknowledgment number is valid. • PSH: requests the receiver to deliver the data to the application upon arrival and not buffer it until a full buffer has been received. • RST: In general, this bit set indicates a problem in the connection. • SYN: used to establish a connection • FIN: specifies that the sender has no more data to transmit.
49 50	msb	TCP Window Size	Variable	• indicates how many bytes may be sent starting at the byte acknowledged.
51 52	msb	TCP Checksum	Variable	• checksums the TCP header, data, and the conceptual TCP pseudoheader.
53 54	msb	TCP Urgent Pointer		• Indicates a byte offset from the current sequence number at which urgent data are to be found

Table 32. TCP Header Format. [6].

G. AAAV SPECIFIC DATA FORMAT

FDDI Header (14 Bytes)	IP Header (20 Bytes)	TCP Header (20 Bytes)	AAAV Data (up to 4440 Bytes)	FDDI Trailer (6 Bytes)

The format for the AAAV Specific Data field is identified in Table 33.

Byte	Bit Order	Field ID	Values (decimal)	Description
55	msb	Packet Update Identification Tag (PUIT)	0-255	• Ring-tailed counter • Provides information to the application software concerning data modifications
56 57 ... n-6[12]	msb	Data	Variable	• Data specific to the message being sent

Table 33. AAAV Specific Data Field Format. [6]

APPENDIX C. AN OVERVIEW OF OPNET

This appendix provides a brief overview of the Optimized Network Engineering Tools (OPNET) modeling and simulation software package. This appendix reviews the OPNET hierarchy, the concept of OPNET's simulation kernel, the employment of interrupts within an OPNET simulation, the definition of an OPNET process module, and the various tools available within OPNET. In addition, the objective of this appendix is to familiarize the reader with the basic functions of OPNET. The reader requiring a more detailed or extensive understanding of OPNET's modeling software package is referred to the 12 volume set of the OPNET user manuals.

A. THE OPNET HIERARCHY

OPNET is a windows-based program that uses windows, dialog boxes, toolbar buttons, and scroll bars, while emphasizing use of the workstation mouse for input whenever possible. OPNET is a comprehensive engineering tool capable of modeling large communications networks with detailed protocol modeling and performance analysis. OPNET is hierarchical in that it implements models in three levels: the network level, the node level, and the process level - also referred to as *modeling domains*. The hierarchy is delineated in Figure 33.

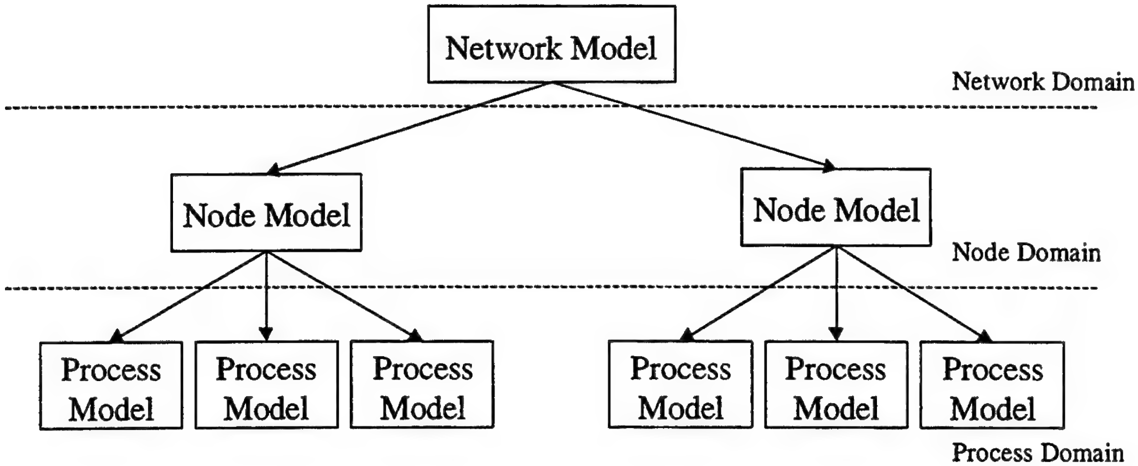

Figure 33. OPNET Hierarchical Organization.

1. Network Domain

The network domain defines the bounds of the network. It describes the communications network in terms of subnetworks, nodes, and links. Figure 10 shows the AAAV-P network model used in this thesis, and is repeated here as Figure 34. It consists of four nodes and numerous point-to-point links. Network models can also employ bus links and radio links.

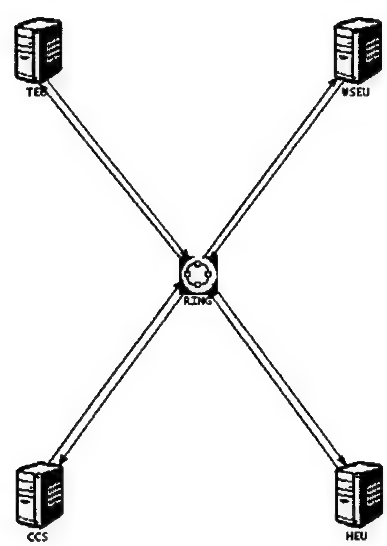

Figure 34. AAAV-P High Speed Data Bus Network Model.

2. Node Domain

The node domain describes the internal architecture in terms of functional elements and data flow. A node model is defined by connecting various modules with packet streams and statistic wires. The connections between modules allow packets and status information to be exchanged between modules and are the primary communication link between the different modules within the node. Each module placed in a node serves a specific purpose such as generating packets, queuing packets, processing packets, or transmitting and receiving packets. Figure 11 offers a graphical view of the TEU node model implemented in this thesis, and is repeated here as Figure 35. As shown, it consists of a transmitter module, a receiver module, two queues (*mac* and *ip*), and six general process modules. The *teu_msg_gen* and *teu_msg_rcvr* are two examples of generic processes that the designer has full flexibility in designing. Implementing these generic processes is commonly where the full modeling capability of OPNET is realized. Communication between the node modules is enabled using standard interface control information (ICI) formats.

3. Process Domain

The process domain defines the behavior of the functional elements within the node domain. Each node model has an underlying process model. The behavior is specified using finite state machines and a high-level language (C). Process models can represent the logic of communications hardware, network protocols, distributed algorithms, or high-level client-server processes. The OPNET software provides a library containing numerous standard models (such as ATM, FDDI, and Ethernet) as well as models contributed by OPNET users. However, at the core of most OPNET simulations are user-defined process models. Processes are usually invoked into execution when the simulation reaches the time

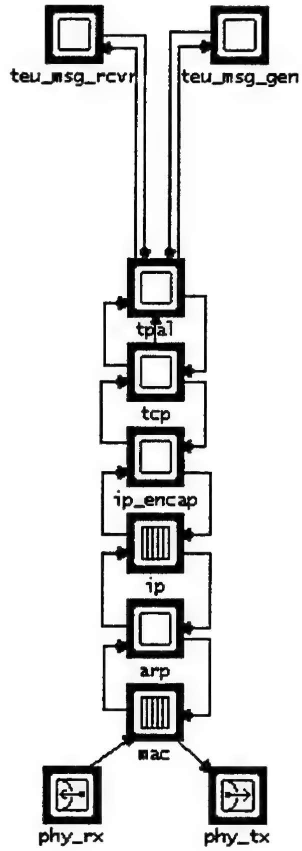

Figure 35. TEU Node Model.

a specific process is scheduled to be invoked or when a specific process receives an "interrupt". The following section provides information on interrupts and the simulation kernel. Thereafter, a more detailed review of defining a process is provided.

B. INTERRUPTS AND THE SIMULATION KERNEL

OPNET's simulation kernel (SK) can be considered the master scheduler and the master clock within OPNET simulations. OPNET is a discrete time, event driven system. The time axis within an OPNET simulation is referred to as sim_time(). A sim_time() equal to 0.0 indicates the beginning of the simulation. As simulations are executed within OPNET, sim_time() advances and a series of events are scheduled to occur at specific times throughout the simulation. Typical events might include the sending of a packet every X

number of seconds, sampling a queue size periodically, decrementing a counter, or a number of other events that the designer has defined. The SK is responsible for scheduling these various events and ensuring they are executed at the scheduled sim_time().

Scheduled events are typically executed by the invocation of different processes that define the process modules. Processes are typically invoked by one of two methods: 1) sim_time() reaches the time the process is scheduled to be invoked and the process is automatically invoked into execution, or 2) OPNET's SK alerts the process by sending an interrupt to the respective process via an input stream. The interrupt delivered to the process invokes the process into execution, and the functions of the process are carried out according to the definition of the process. From the designer's perspective, interrupts are the most common method of invoking processes.

The focus of modeling in OPNET is in defining each process model that supports and implements each node model. The process models are implemented as Finite State Machines (FSMs). An FSM is defined by completing a state transition diagram that defines the various states of the process and the transitions that interconnect each state. The process of generating and sending messages from the TEU node is executed by the invocation of the *teu_msg_gen* process module as portrayed in Figure 13. Figure 36 shows the same process as viewed from within the OPNET graphical environment.

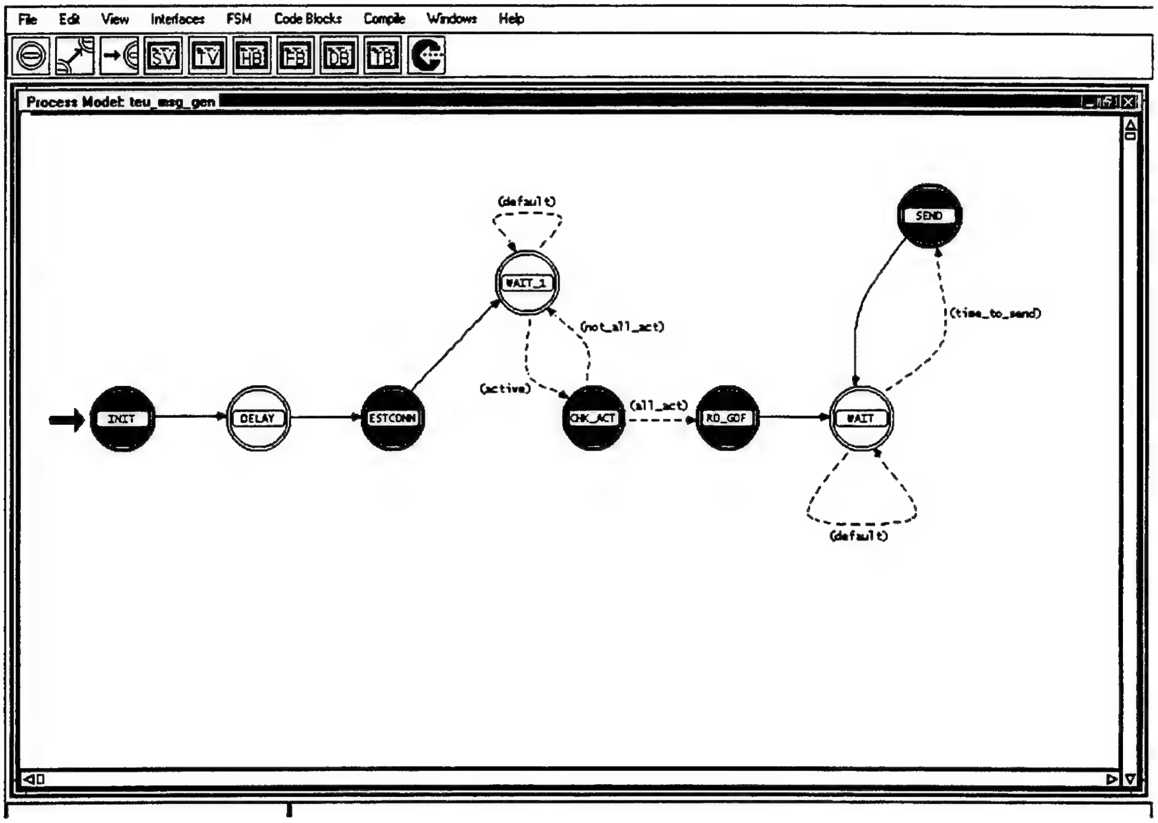

Figure 36. OPNET View of the *teu_msg_gen* Process Model.

C. DEFINING THE PROCESS

1. OPNET'S Graphical User Interface (GUI)

As seen in Figure 36, the *teu_msg_gen* process consists of eight states. The actions performed within each state are dictated by the designer's source code (written in C). In completing the source code for a process, the icon buttons at the top of the window play an important role; a close up view of these icon buttons is shown in Figure 37. The three left most buttons are used to identify new states and transitions between the states. The SV, TV, HB, FB, DB, and TB buttons will invoke the OPNET editor in which the designer enters the relevant source code.

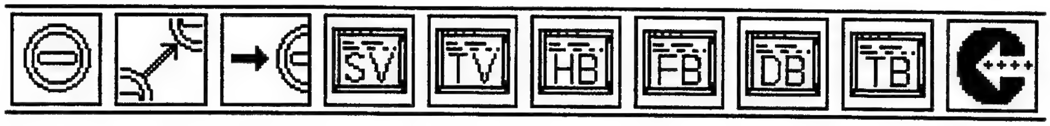

Figure 37. OPNET Process Editor Icon Buttons.

The SV icon button is used to define the *state variables* used in the process. Similarly, the TV icon button defines the *temporary variables* of the process. The HB, FB, DB, TB icon buttons define the *header block*, *function block*, *diagnostic block*, and *termination block* respectively for the process. The icon button on the far right compiles the process model code.

The *header block* is similar to a C++ include.h file. #include <> statements and #define statements are likely to be coded in the header block. Temporary variables are declared in the *temporary variables* block and are not persistent variables. They exist only during the current invocation of the process. State variables are declared in the *state variables* block. They are persistent and retain their value from one invocation of a process to the next. For example, the *aaav_msg_rcvr* process shown in Figure 12, and repeated below in Figure 38, is invoked every time a packet is received at a network node. Any defined temporary variables are persistent only for the processing of this specific packet. Any state variables are persistent for all packets, as state variables are persistent throughout the simulation. A loop counter is an example of a common temporary variable. State variables are similar to static variables in C. Accumulators are a common example of state variables.

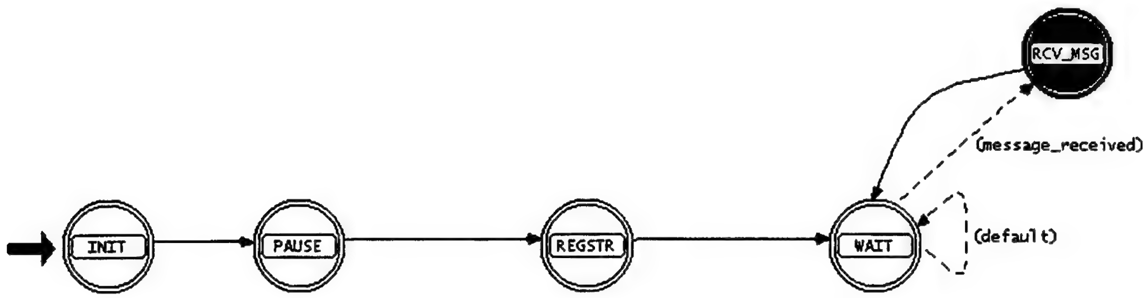

Figure 38. State Transition Diagram of the *aaav_msg_rcvr*.

A *function block* is utilized by the designer to define the various functions used within the process. The functions are typically called from within a state. However, functions may also be called from within another function. A common approach is to minimize the code within the states so that they are defined primarily by a series of function calls, and to off-load the bulk of the processing to the various functions defined in the function block. The *termination block* is invoked just before the process is destroyed. The *diagnostic block* contains user written C statements that send diagnostic information to the standard output device during a simulation.

2. Forced and Unforced States

There is significance to the different color shading of the states shown in Figure 38. States within OPNET are either *forced* or *unforced* and are shown in Figure 39. Unforced states are green on a color display (transparent on hardcopy), while forced states are red (opaque). Each state has an *enter executive* that is executed upon entering the state and an *exit executive* that is executed when leaving the state. A forced state is synonymous to an autonomous instruction; it does not allow any interruptions during its execution. A forced state controls the simulation until it has completed all of the actions contained in both its

enter and exit executives; it cannot be interrupted by the SK. After a forced state has completed its exit executive, the process proceeds immediately to the next state and executes its enter executives. An unforced state, on the other hand, returns control of the simulation to the SK once the process has executed its enter executives. A process that is in an unforced state is merely waiting (blocked) for another invocation call from the SK.

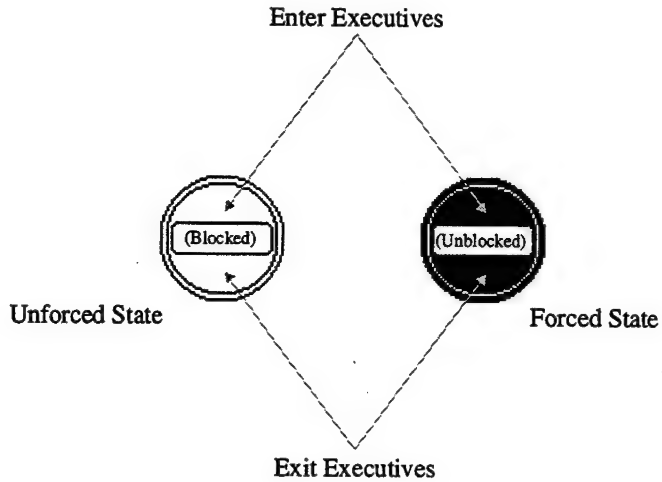

Figure 39. Graphical Representation of Forced and Unforced States.

Once the SK has regained control of the simulation from the process (indicating the process is in an unforced state), the SK is free to invoke other processes of the simulation. The key difference between forced and unforced states is that there is no time lapse between entering and exiting a forced state. All actions within forced states are essentially executed at the same time. With an unforced state, the time between entering and exiting the state is determined by the overall simulation and the different processes involved. No time lapse is incurred when executing one of the executives. In contrast, there is a time lapse between the moment the state is entered and the time at which it is exited.

As an example of the above discussion, consider the *aaav_msg_rcvr* process defined by the state transition diagram of Figure 38. The opaque RCV_MSG state is a forced state,

while the remaining transparent states indicate that they are unforced states. At the beginning of the simulation, the INIT state is evaluated and initialization of the process is performed (Chapter III outlines the actions performed in this particular INIT state). As the INIT state is an unforced state, control of the simulation returns to the SK after the enter executive has been executed. No time lapse was incurred during the initialization, and sim_time() = t_0 indicating the simulation time clock (sim_time()) has not advanced. After the INIT enter executive has been completed, the state is blocked and must wait to receive an interrupt in order to execute the exit executive and proceed to the PAUSE state.

Upon regaining control of the simulation from the INIT process, the SK checks the event list. The SK determines the next scheduled event and the time that this event is scheduled to occur. The SK will then notify the relevant process at the appropriate time. The SK's notification to a process is typically performed by delivering an interrupt to the respective process.

3. A Packet Flow Example

Consider the flow of a packet from transmission to reception. At some point in time, the message generator process transmits a packet. The packet is evaluated by the pipeline stages, and the SK determines the time at which the packet is scheduled to arrive at the receiver, $t_{arrival}$. The SK then schedules a stream interrupt (OPC_INTRPT_STREAM) to be delivered to this process at this same arrival time, $t_{arrival}$. At sim_time() = $t_{arrival}$, the SK delivers a stream interrupt to the message receiver process. This delivered interrupt will invoke this process into execution (assuming the process is in an unforced state).

We have seen where the *aaav_msg_rcvr* process performed the initialization functions at the beginning of the simulation. Assume the process has completed the INIT,

PAUSE, and REGSTR states and has transitioned to the unforced WAIT state. At this point, when the SK delivers an interrupt to this process (sim_time() = $t_{arrival}$), the message receiver process is "resting" idly in the unforced WAIT state. The definition within the header block of this process model is shown in Figure 14.

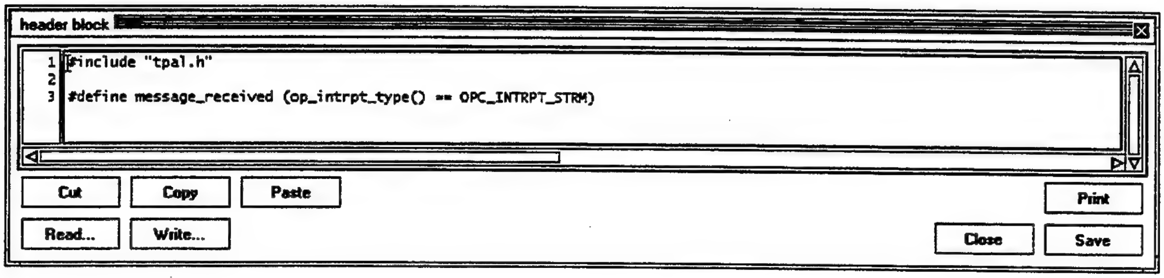

Figure 40. Header Block Definitions for the *aaav_msg_rcvr* Process Model.

The SK delivered interrupt invokes the execution of this aaav_msg_rcvr process. Thus at sim_time() = $t_{arrival}$, this message receiver process begins execution. Upon execution, the process determines that message_received evaluates as TRUE (the interrupt just delivered is equal to a *stream type* interrupt) and will thus transition to the RCV_MSG state.

As the RCV_MSG state is a forced state, it will execute the code defined within the RCV_MSG state without interruption from the SK.

D. OPNET TOOLS

All of the design related to a particular network model is performed within an OPNET tool window. The OPNET program contains nine tools.

1. Model Development Tools

- Network Editor - The Network Editor is one of the primary development tools and is used to define the construct of the network model. It

provides the resources necessary to model the high-level components of the network.

- Node Editor - The Node Editor is another primary development tool and is used to create models of nodes. It is used to define the internal functioning of nodes.
- Process Editor - The Processor Editor is the third primary development tool. It is utilized to define state models for the processes that run in processor and queue models. Definition of process models is accomplished through the use of state transitions diagrams.
- Packet Format Editor - The Packet Format Editor allows the designer to define the internal structure of a packet as a set of fields.
- Parameter Editor - The Parameter Editor is used in conjunction with the process and node model development to define special model structures. The Parameter Editor has five modes. Each mode is used to define a particular construct such as ICI Formats, Probability Density Functions, Link Models, Modulation Functions, and Antenna Patterns.

2. Simulation Execution Tools

- Probe Editor - The Probe Editor allows the designer to attach probes to the points of interest in a model prior to simulation execution.
- Simulation Tool - The Simulation tool allows the specification of a simulation sequence which uses particular input and output options.

3. Results Analysis Tools

- Analysis Tool - The Analysis Tool presents statistics gathered during simulations in the form of two dimensional graphs or text listings. The information collected during a simulation can be viewed directly or processed by filters.
- Filter Editor - The Filter Editor is used to define filters to mathematically process, reduce, or combine statistical data.

[25] [26]

APPENDIX D. OPNET CODE FOR MSG_GEN MODULES

This appendix provides the OPNET source code for the *ccs_msg_gen*, *heu_msg_gen*, *teu_msg_gen*, and *wseu_msg_gen* node modules. The code includes comments when appropriate.

The *msg_gen* state diagram is shown in Figure 13 and is repeated below in Figure 41.

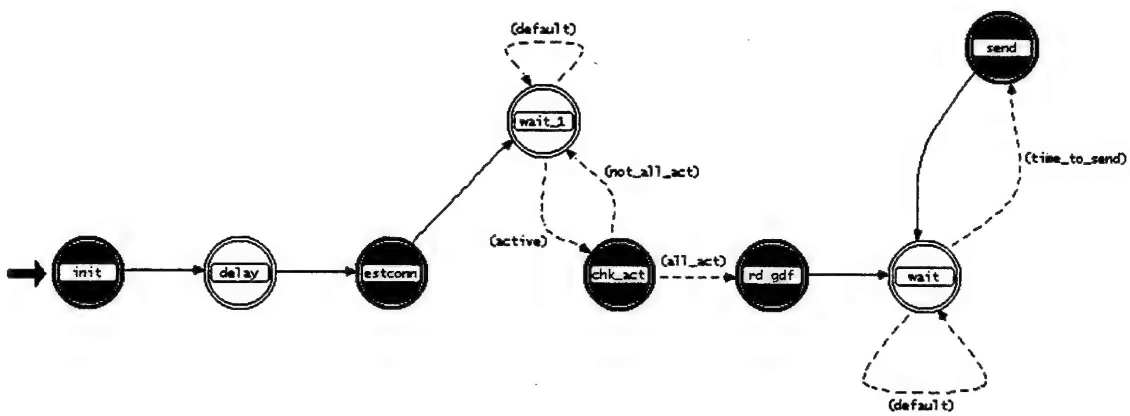

Figure 41. State Transition Diagram of the *msg_gen* Modules.

OPNET CODE FOR CCS_MSG_GEN

Header Block
Deborah G. Peyton
May 1999

```
#include "tpal.h"
#include <string.h>
#include <stdlib.h>

#define NEEDCONN (op_intrpt_type() == OPC_INTRPT_SELF)
#define active ((op_intrpt_type() == OPC_INTRPT_REMOTE) &&
(op_intrpt_code() == TPALC_EV_CONF_OPEN))
#define not_all_act (act_connect <= 2)
#define all_act (act_connect == 3)
#define time_to_send (op_intrpt_type() == OPC_INTRPT_SELF)

/* define the structure for the message parameters */
typedef struct
      {
      char msg_name[30];
      char destination[10];
      int size_of_data;
      int frequency;
      int variance;
      char distribution[15];
      double send_rate;
      } Msg_to_send;
```

State Variables Block

```
Objid               \tpal_objid;
Ici *               \ici_ptr_wseu;
Ici *               \ici_ptr_heu;
Ici *               \ici_ptr_teu;

/* the number of messages in the GDF */
int                 \num_msg;

Msg_to_send *       \msg_to_send;
int                 \intrpt_code;

/* counter */
int                 \x;

/* the number of active TCP connections established */
int                 \act_connect;

unsigned            \seed;
Distribution *      \gen_dist;
double              \rand_start_time;
```

Temporary Variables Block

```
double              start_time;
Packet *            pk_ptr;

List *              gdf_list_ptr;
```

```
char *              gdf_entry_string;
List *              gdf_entry_ptr;
int                 gdf_entry_size;
char *              gdf_element_string;
int                 i,j;

char *              temp_msg_name;
char *              temp_destination;
char *              temp_distribution;
```

INIT STATE

```
/*** Enter executive ***/

op_ima_obj_attr_get (op_id_self(), "Application Start Time",
&start_time);

/* initialize variables */
act_connect = 0;
seed = 350;
srand(seed);
```

DELAY STATE

```
/*** Enter executive ***/

/*create delay to allow all servers to register before tcp connections
are established */
op_intrpt_schedule_self (op_sim_time() + 1, 10000);
```

ESTCONN STATE

```
/*** Exit executive ***/

/* establish connection with WSEU */
ici_ptr_wseu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_wseu);
op_ici_attr_set (ici_ptr_wseu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_wseu, "Service", "FDDI - source CCS");
op_ici_attr_set (ici_ptr_wseu, "Remote Port", 4);
op_ici_attr_set (ici_ptr_wseu, "Local Port", 5);
op_ici_attr_set (ici_ptr_wseu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_wseu, "Remote Address", "WSEU");
tpal_objid = op_topo_assoc (op_id_self(), OPC_TOPO_ASSOC_IN,
OPC_OBJMTYPE_MODULE, 0);
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with HEU */
ici_ptr_heu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_heu);
op_ici_attr_set (ici_ptr_heu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_heu, "Service", "FDDI - source CCS");
op_ici_attr_set (ici_ptr_heu, "Remote Port", 4);
op_ici_attr_set (ici_ptr_heu, "Local Port", 6);
op_ici_attr_set (ici_ptr_heu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_heu, "Remote Address", "HEU");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with TEU */
ici_ptr_teu = op_ici_create ("tpal_req");
```

```
op_ici_install (ici_ptr_teu);
op_ici_attr_set (ici_ptr_teu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_teu, "Service", "FDDI - source CCS");
op_ici_attr_set (ici_ptr_teu, "Remote Port", 4);
op_ici_attr_set (ici_ptr_teu, "Local Port", 7);
op_ici_attr_set (ici_ptr_teu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_teu, "Remote Address", "TEU");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);
```

WAIT_1 STATE

```
/*** Enter executive ***/

/* this state waits for all three tpal interfaces to be established */
```

CHK_ACT STATE

```
/*** Enter executive ***/

act_connect++;


/*** Exit executive ***/

/* proceed to SEND state only if all three tpal connections have been
opened */
/* otherwise, return to WAIT_1 state */
```

RD_GDF STATE

```
/*** Enter executive ***/

/* read the general data file (GDF) */
gdf_list_ptr = op_prg_gdf_read ("fddi_ccs_s1-3");

/* get the number of messages in the GDF file */
num_msg = op_prg_list_size (gdf_list_ptr);

/* create a structure and allocate mem for each message */
msg_to_send = (Msg_to_send *) op_prg_mem_alloc
(sizeof(Msg_to_send)*num_msg);

/* get each line in the GDF file */
for (i = 0; i < num_msg; i++)
      {
      /* get the ith line in the GDF file */
      gdf_entry_string = op_prg_list_access (gdf_list_ptr, i);

      /* decompose the string into its components */
      gdf_entry_ptr = op_prg_str_decomp (gdf_entry_string, ",");

      /* get the number of fields in the line */
      gdf_entry_size = op_prg_list_size (gdf_entry_ptr);

      if (gdf_entry_size < 7)     /* ensure valid # of fields */
            {
            /* break out the fields into structure */
            temp_msg_name = (char *) (op_prg_list_access (gdf_entry_ptr,
0));
            strcpy (msg_to_send[i].msg_name, temp_msg_name);
```

```
            temp_destination = (char *) (op_prg_list_access
(gdf_entry_ptr, 1));
            strcpy (msg_to_send[i].destination, temp_destination);
            msg_to_send[i].size_of_data = atoi (op_prg_list_access
(gdf_entry_ptr, 2));
            msg_to_send[i].frequency = atoi (op_prg_list_access
(gdf_entry_ptr, 3));
            msg_to_send[i].variance = atoi (op_prg_list_access
(gdf_entry_ptr, 4));
            temp_distribution = (char *) (op_prg_list_access
(gdf_entry_ptr, 5));
            strcpy (msg_to_send[i].distribution, temp_distribution);
            }
      }

/* deallocate the table list and contents */
op_prg_list_free (gdf_list_ptr);
op_prg_mem_free (gdf_list_ptr);

/* schedule interrupts for each message with op_intrpt_code = x*/
for (x = 0; x < num_msg; x++)
      {
      /* start all interrupts randomly within the first 50 sec */
      rand_start_time = ((rand() % 50) + op_dist_uniform(2));

      /* schedule first message to be sent at the random start time */
      op_intrpt_schedule_self ((op_sim_time () + rand_start_time), x);
      }
```

WAIT STATE

```
/*** Enter executive ***/

/* this state does nothing except wait. */
/* when it is time for a message to be sent, the process proceeds to the
next state */


/*** Exit executive ***/

intrpt_code = op_intrpt_code ();  /* store original op_intrpt_code */
```

SEND STATE

```
/*** Enter executive ***/

pk_ptr = op_pk_create_fmt ("AAAV-FDDI_pk");

/* set size of packet */
op_pk_total_size_set (pk_ptr, msg_to_send[intrpt_code].size_of_data);

//op_pk_nfd_set (pk_ptr, "PUIT", 0);          /* place holders for future
use */
//op_pk_nfd_set (pk_ptr, "Data", "test");

if (strcmp (msg_to_send[intrpt_code].destination, "HEU") == 0)
      {
      op_ici_install (ici_ptr_heu);
      }

else if (strcmp (msg_to_send[intrpt_code].destination, "TEU") == 0)
```

```
        {
        op_ici_install (ici_ptr_teu);
        }

else if (strcmp (msg_to_send[intrpt_code].destination, "WSEU") == 0)
        {
        op_ici_install (ici_ptr_wseu);
        }

else
        {
        /* must be a type-o in the GDF */
        printf ("invalid destination in CCS file on line %d\n",
intrpt_code);
        op_ici_install (ici_ptr_heu);    /* default to send to HEU */
        }

/* send the packet */
op_pk_send (pk_ptr, 0);
op_ici_install (OPC_NIL);


/*** Exit executive ***/

/* to make the message generator process truly random, we need to set the
next time this message is sent at another randomly generated time */

do
        {
        if (strcmp(msg_to_send[intrpt_code].distribution, "CONSTANT") == 0)
                {
                msg_to_send[intrpt_code].send_rate =
msg_to_send[intrpt_code].frequency;
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "UNIFORM")
== 0)
                {
                msg_to_send[intrpt_code].send_rate = op_dist_uniform
(msg_to_send[intrpt_code].frequency);
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "NORMAL") ==
0)
                {
                gen_dist = op_dist_load ("normal",
msg_to_send[intrpt_code].frequency, msg_to_send[intrpt_code].variance);
                msg_to_send[intrpt_code].send_rate = op_dist_outcome
(gen_dist);
                }
} while (msg_to_send[intrpt_code].send_rate <= 0);    /* repeat until the
send_rate is not equal to zero */

/* schedule another interrupt at this frequency */
op_intrpt_schedule_self (op_sim_time () + (double) 1.0/ ((double)
msg_to_send[intrpt_code].send_rate * 1/3600), intrpt_code);
```

OPNET CODE FOR HEU_MSG_GEN

Header Block
Deborah G. Peyton
May 1999

```
#include "tpal.h"
#include <string.h>
#include <stdlib.h>

#define NEEDCONN (op_intrpt_type() == OPC_INTRPT_SELF)
#define active ((op_intrpt_type() == OPC_INTRPT_REMOTE) &&
(op_intrpt_code() == TPALC_EV_CONF_OPEN))
#define not_all_act (act_connect <= 2)
#define all_act (act_connect == 3)
#define time_to_send (op_intrpt_type() == OPC_INTRPT_SELF)

/* define the structure for the message parameters */
typedef struct
      {
      char msg_name[30];
      char destination[10];
      int size_of_data;
      int frequency;
      int variance;
      char distribution[15];
      double send_rate;
      } Msg_to_send;
```

State Variables Block

```
Objid                  \tpal_objid;
Ici *                  \ici_ptr_wseu;
Ici *                  \ici_ptr_teu;
Ici *                  \ici_ptr_ccs;

/* the number of messages in the GDF */
int                    \num_msg;

Msg_to_send *          \msg_to_send;
int                    \intrpt_code;

/* counter */
int                    \x;

/* the number of active TCP connections established */
int                    \act_connect;

unsigned               \seed;
Distribution *         \gen_dist;
double                 \rand_start_time;
```

Temporary Variables Block

```
double                 start_time;
Packet *               pk_ptr;

List *                 gdf_list_ptr;
```

```
char *              gdf_entry_string;
List *              gdf_entry_ptr;
int                 gdf_entry_size;
char *              gdf_element_string;
int                 i,j;

char *              temp_msg_name;
char *              temp_destination;
char *              temp_distribution;
```

INIT STATE

```
/*** Enter executive ***/

op_ima_obj_attr_get (op_id_self(), "Application Start Time",
&start_time);

/* initialize variables */
act_connect = 0;
seed = 648;
srand(seed);
```

DELAY STATE

```
/*** Enter executive ***/

/* create delay to allow all servers to register before tcp connections
are established */
op_intrpt_schedule_self (op_sim_time() + 1, 10000);
```

ESTCONN STATE

```
/*** Exit executive ***/

/* establish connection with WSEU */
ici_ptr_wseu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_wseu);
op_ici_attr_set (ici_ptr_wseu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_wseu, "Service", "FDDI - source HEU");
op_ici_attr_set (ici_ptr_wseu, "Remote Port", 3);
op_ici_attr_set (ici_ptr_wseu, "Local Port", 5);
op_ici_attr_set (ici_ptr_wseu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_wseu, "Remote Address", "WSEU");
tpal_objid = op_topo_assoc (op_id_self(), OPC_TOPO_ASSOC_IN,
OPC_OBJMTYPE_MODULE, 0);
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with TEU */
ici_ptr_teu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_teu);
op_ici_attr_set (ici_ptr_teu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_teu, "Service", "FDDI - source HEU");
op_ici_attr_set (ici_ptr_teu, "Remote Port", 3);
op_ici_attr_set (ici_ptr_teu, "Local Port", 6);
op_ici_attr_set (ici_ptr_teu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_teu, "Remote Address", "TEU");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with CCS */
ici_ptr_ccs = op_ici_create ("tpal_req");
```

```
op_ici_install (ici_ptr_ccs);
op_ici_attr_set (ici_ptr_ccs, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_ccs, "Service", "FDDI - source HEU");
op_ici_attr_set (ici_ptr_ccs, "Remote Port", 3);
op_ici_attr_set (ici_ptr_ccs, "Local Port", 7);
op_ici_attr_set (ici_ptr_ccs, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_ccs, "Remote Address", "CCS");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);
```

WAIT_1 STATE

```
/*** Enter executive ***/

/* this state waits for all three tpal interfaces to be established */
```

CHK_ACT STATE

```
/*** Enter executive ***/

act_connect++;


/*** Exit executive ***/

/* proceed to SEND state only if all three tpal connections have been
opened */
/* otherwise, return to WAIT_1 state */
```

RD_GDF STATE

```
/*** Enter executive ***/

/* read the general data file (GDF) */
gdf_list_ptr = op_prg_gdf_read ("fddi_heu_s1");

/* get the number of messages in the GDF file */
num_msg = op_prg_list_size (gdf_list_ptr);

/* create a structure and allocate mem for each message */
msg_to_send = (Msg_to_send *) op_prg_mem_alloc
(sizeof(Msg_to_send)*num_msg);

/* get each line in the GDF file */
for (i = 0; i < num_msg; i++)
	{
	/* get the ith line in the GDF file */
	gdf_entry_string = op_prg_list_access (gdf_list_ptr, i);

	/* decompose the string into its components */
	gdf_entry_ptr = op_prg_str_decomp (gdf_entry_string, ",");

	/* get the number of fields in the line */
	gdf_entry_size = op_prg_list_size (gdf_entry_ptr);

	if (gdf_entry_size < 7)    /* ensure valid # of fields */
		{
		/* break out the fields into structure */
		temp_msg_name = (char *) (op_prg_list_access (gdf_entry_ptr,
0));
		strcpy (msg_to_send[i].msg_name, temp_msg_name);
```

```
            temp_destination = (char *) (op_prg_list_access
(gdf_entry_ptr, 1));
            strcpy (msg_to_send[i].destination, temp_destination);
            msg_to_send[i].size_of_data = atoi (op_prg_list_access
(gdf_entry_ptr, 2));
            msg_to_send[i].frequency = atoi (op_prg_list_access
(gdf_entry_ptr, 3));
            msg_to_send[i].variance = atoi (op_prg_list_access
(gdf_entry_ptr, 4));
            temp_distribution = (char *) (op_prg_list_access
(gdf_entry_ptr, 5));
            strcpy (msg_to_send[i].distribution, temp_distribution);
            }
      }

/* deallocate the table list and contents */
op_prg_list_free (gdf_list_ptr);
op_prg_mem_free (gdf_list_ptr);

/* schedule interrupts for each message with op_intrpt_code = x*/
for (x = 0; x < num_msg; x++)
      {
      /* start all interrupts randomly within the first 50 sec */
      rand_start_time = ((rand() % 50) + op_dist_uniform(2));

      /* schedule first message to be sent at the random start time */
      op_intrpt_schedule_self ((op_sim_time () + rand_start_time), x);
      }
```

WAIT STATE

```
/*** Enter executive ***/

/* this state does nothing except wait. */
/* when it is time for a message to be sent, the process proceeds to the
next state */


/*** Exit executive ***/

intrpt_code = op_intrpt_code ();  /* store original op_intrpt_code */
```

SEND STATE

```
/*** Enter executive ***/

pk_ptr = op_pk_create_fmt ("AAAV-FDDI_pk");

/* set size of packet */
op_pk_total_size_set (pk_ptr, msg_to_send[intrpt_code].size_of_data);

//op_pk_nfd_set (pk_ptr, "PUIT", 0);      /* place holders for future use
*/
//op_pk_nfd_set (pk_ptr, "Data", "test");

if (strcmp (msg_to_send[intrpt_code].destination, "TEU") == 0)
      {
      op_ici_install (ici_ptr_teu);
      }

else if (strcmp (msg_to_send[intrpt_code].destination, "CCS") == 0)
```

```
        {
        op_ici_install (ici_ptr_ccs);
        }

else if (strcmp (msg_to_send[intrpt_code].destination, "WSEU") == 0)
        {
        op_ici_install (ici_ptr_wseu);
        }

else
        {
        /* must be a type-o in the GDF */
        printf ("invalid destination in HEU file on line %d\n",
intrpt_code);
        op_ici_install (ici_ptr_teu);    /* default to send to TEU */
        }

/* send the packet */
op_pk_send (pk_ptr, 0);
op_ici_install (OPC_NIL);


/*** Exit executive ***/

/* to make the message generator process truly random, we need to set the
next time this message is sent at another randomly generated time */

do
        {
        if (strcmp(msg_to_send[intrpt_code].distribution, "CONSTANT") == 0)
                {
                msg_to_send[intrpt_code].send_rate =
msg_to_send[intrpt_code].frequency;
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "UNIFORM")
== 0)
                {
                msg_to_send[intrpt_code].send_rate = op_dist_uniform
(msg_to_send[intrpt_code].frequency);
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "NORMAL") ==
0)
                {
                gen_dist = op_dist_load ("normal",
msg_to_send[intrpt_code].frequency, msg_to_send[intrpt_code].variance);
                msg_to_send[intrpt_code].send_rate = op_dist_outcome
(gen_dist);
                }
} while (msg_to_send[intrpt_code].send_rate <= 0);     /* repeat until
the send rate is not equal to zero */

/* schedule another interrupt at this frequency */
op_intrpt_schedule_self (op_sim_time () + (double) 1.0/ ((double)
msg_to_send[intrpt_code].send_rate * 1/3600), intrpt_code);
```

OPNET CODE FOR TEU_MSG_GEN

Header Block
Deborah G. Peyton
May 1999

```
#include "tpal.h"
#include <string.h>
#include <stdlib.h>

#define NEEDCONN (op_intrpt_type() == OPC_INTRPT_SELF)
#define active ((op_intrpt_type() == OPC_INTRPT_REMOTE) &&
(op_intrpt_code() == TPALC_EV_CONF_OPEN))
#define not_all_act (act_connect <= 2)
#define all_act (act_connect == 3)
#define time_to_send (op_intrpt_type() == OPC_INTRPT_SELF)

/* define the structure for the message parameters */
typedef struct
        {
        char msg_name[30];
        char destination[10];
        int size_of_data;
        int frequency;
        int variance;
        char distribution[15];
        double send_rate;
        } Msg_to_send;
```

State Variables Block

```
Objid               \tpal_objid;
Ici *               \ici_ptr_wseu;
Ici *               \ici_ptr_heu;
Ici *               \ici_ptr_ccs;

/* the number of messages contained in the GDF */
int                 \num_msg;

Msg_to_send *       \msg_to_send;
int                 \intrpt_code;

/* counter */
int                 \x;

/* the number of active TCP connections established */
int                 \act_connect;

unsigned            \seed;

Distribution *      \gen_dist;
double              \rand_start_time;
```

Temporary Variables Block

```
double              start_time;
Packet *            pk_ptr;
```

```
List *              gdf_list_ptr;
char *              gdf_entry_string;
List *              gdf_entry_ptr;
int                 gdf_entry_size;
char *              gdf_element_string;
int                 i,j;

char *              temp_msg_name;
char *              temp_destination;
char *              temp_distribution;
```

INIT STATE

```
/*** Enter executive ***/

op_ima_obj_attr_get (op_id_self(), "Application Start Time",
&start_time);

/* initialize variables */
act_connect = 0;
seed = 128;

srand(seed);
```

DELAY STATE

```
/*** Enter executive ***/

/* create delay to allow all servers to register before tcp connections
are established */
op_intrpt_schedule_self (op_sim_time() + 1, 10000);
```

ESTCONN STATE

```
/*** Exit executive ***/

/* establish connection with WSEU */
ici_ptr_wseu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_wseu);
op_ici_attr_set (ici_ptr_wseu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_wseu, "Service", "FDDI - source TEU");
op_ici_attr_set (ici_ptr_wseu, "Remote Port", 1);
op_ici_attr_set (ici_ptr_wseu, "Local Port", 5);
op_ici_attr_set (ici_ptr_wseu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_wseu, "Remote Address", "WSEU");
tpal_objid = op_topo_assoc (op_id_self(), OPC_TOPO_ASSOC_IN,
OPC_OBJMTYPE_MODULE, 0);
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with HEU */
ici_ptr_heu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_heu);
op_ici_attr_set (ici_ptr_heu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_heu, "Service", "FDDI - source TEU");
op_ici_attr_set (ici_ptr_heu, "Remote Port", 1);
op_ici_attr_set (ici_ptr_heu, "Local Port", 6);
op_ici_attr_set (ici_ptr_heu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_heu, "Remote Address", "HEU");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);
```

```
/* establish connection with CCS */
ici_ptr_ccs = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_ccs);
op_ici_attr_set (ici_ptr_ccs, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_ccs, "Service", "FDDI - source TEU");
op_ici_attr_set (ici_ptr_ccs, "Remote Port", 1);
op_ici_attr_set (ici_ptr_ccs, "Local Port", 7);
op_ici_attr_set (ici_ptr_ccs, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_ccs, "Remote Address", "CCS");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);
```

WAIT_1 STATE

```
/*** Enter executive ***/

/* this state waits for all three tpal interfaces to be established */
```

CHK_ACT STATE

```
/*** Enter executive ***/

act_connect++;


/*** Exit executive ***/

/* proceed to SEND state only if all three tpal connections have been
opened */
/* otherwise, return to WAIT_1 state */
```

RD_GDF STATE

```
/*** Enter executive ***/

/* read the general data file (GDF) */
gdf_list_ptr = op_prg_gdf_read ("fddi_teu_all");

/* get the number of messages in the GDF file */
num_msg = op_prg_list_size (gdf_list_ptr);

/* create a structure and allocate mem for each message */
msg_to_send = (Msg_to_send *) op_prg_mem_alloc
(sizeof(Msg_to_send)*num_msg);

/* get each line in the GDF file */
for (i = 0; i < num_msg; i++)
      {
      /* get the ith line in the GDF file */
      gdf_entry_string = op_prg_list_access (gdf_list_ptr, i);

      /* decompose the string into its components */
      gdf_entry_ptr = op_prg_str_decomp (gdf_entry_string, ",");

      /* get the number of fields in line */
      gdf_entry_size = op_prg_list_size (gdf_entry_ptr);

      if (gdf_entry_size < 7)      /* ensure valid # of fields */
            {
            /* break out fields into structure */
```

```
            temp_msg_name = (char *) (op_prg_list_access (gdf_entry_ptr,
0));
            strcpy (msg_to_send[i].msg_name, temp_msg_name);
            temp_destination = (char *) (op_prg_list_access
(gdf_entry_ptr, 1));
            strcpy (msg_to_send[i].destination, temp_destination);
            msg_to_send[i].size_of_data = atoi (op_prg_list_access
(gdf_entry_ptr, 2));
            msg_to_send[i].frequency = atoi (op_prg_list_access
(gdf_entry_ptr, 3));
            msg_to_send[i].variance = atoi (op_prg_list_access
(gdf_entry_ptr, 4));
            temp_distribution = (char *) (op_prg_list_access
(gdf_entry_ptr, 5));
            strcpy (msg_to_send[i].distribution, temp_distribution);
            }
      }

/* deallocate the table list and contents */
op_prg_list_free (gdf_list_ptr);
op_prg_mem_free (gdf_list_ptr);

/* schedule interrupts for each message with op_intrpt_code = x*/
for (x = 0; x < num_msg; x++)
      {
      /* start all interrupts randomly within the first 50 sec */
      rand_start_time = ((rand() % 50) + op_dist_uniform(2));

      /* schedule first message to be sent at the random start time */
      op_intrpt_schedule_self ((op_sim_time () + rand_start_time), x);
      }
```

WAIT STATE

```
/*** Enter executive ***/

/* this state does nothing except wait.*/
/* when it is time for a message to be sent, the process proceeds to the
next state */


/*** Exit executive ***/

intrpt_code = op_intrpt_code ();   /* store original op_intrpt_code */
```

SEND STATE

```
/*** Enter executive ***/

pk_ptr = op_pk_create_fmt ("AAAV-FDDI_pk");

/* set size of packet */
op_pk_total_size_set (pk_ptr, msg_to_send[intrpt_code].size_of_data);

//op_pk_nfd_set (pk_ptr, "PUIT", 0);          /* place holders for future
use */
//op_pk_nfd_set (pk_ptr, "Data", "test");

if (strcmp (msg_to_send[intrpt_code].destination, "HEU") == 0)
      {
      op_ici_install (ici_ptr_heu);
```

```
        }

else if (strcmp (msg_to_send[intrpt_code].destination, "CCS") == 0)
        {
        op_ici_install (ici_ptr_ccs);
        }

else if (strcmp (msg_to_send[intrpt_code].destination, "WSEU") == 0)
        {
        op_ici_install (ici_ptr_wseu);
        }

else
        {
        /* must be type-o in GDF file */
        printf ("invalid destination in TEU file on line %d\n",
intrpt_code);
        op_ici_install (ici_ptr_heu);       /* default to send to HEU */
        }

/* send the packet */
op_pk_send (pk_ptr, 0);
op_ici_install (OPC_NIL);


/*** Exit executive ***/

/* to make the message generator process truly random, we need to set the
next time this message is sent at another randomly generated time */

do
        {
        if (strcmp(msg_to_send[intrpt_code].distribution, "CONSTANT") == 0)
                {
                msg_to_send[intrpt_code].send_rate =
msg_to_send[intrpt_code].frequency;
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "UNIFORM")
== 0)
                {
                msg_to_send[intrpt_code].send_rate = op_dist_uniform
(msg_to_send[intrpt_code].frequency);
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "NORMAL") ==
0)
                {
                gen_dist = op_dist_load ("normal",
msg_to_send[intrpt_code].frequency, msg_to_send[intrpt_code].variance);
                msg_to_send[intrpt_code].send_rate = op_dist_outcome
(gen_dist);
                }
} while (msg_to_send[intrpt_code].send_rate <= 0);     /* repeat until
the send rate is not equal to zero */

/* schedule another interrupt at this frequency */
op_intrpt_schedule_self (op_sim_time () + (double) 1.0/ ((double)
msg_to_send[intrpt_code].send_rate * 1/3600), intrpt_code);
```

OPNET CODE FOR WSEU_MSG_GEN

Header Block
Deborah G. Peyton
May 1999

```
#include "tpal.h"
#include <string.h>
#include <stdlib.h>

#define NEEDCONN (op_intrpt_type() == OPC_INTRPT_SELF)
#define active ((op_intrpt_type() == OPC_INTRPT_REMOTE) &&
(op_intrpt_code() == TPALC_EV_CONF_OPEN))
#define not_all_act (act_connect <= 2)
#define all_act (act_connect == 3)
#define time_to_send (op_intrpt_type() == OPC_INTRPT_SELF)

/* define the structure for the message parameters */
typedef struct
      {
      char msg_name[30];
      char destination[10];
      int size_of_data;
      int frequency;
      int variance;
      char distribution[15];
      double send_rate;
      } Msg_to_send;
```

State Variables Block

```
Objid                   \tpal_objid;
Ici *                   \ici_ptr_teu;
Ici *                   \ici_ptr_heu;
Ici *                   \ici_ptr_ccs;

/* the number of messages contained in the GDF */
int                     \num_msg;

Msg_to_send *           \msg_to_send;
int                     \intrpt_code;

/* counter */
int                     \x;

/* the number of active TCP connections established */
int                     \act_connect;

unsigned                \seed;

Distribution *          \gen_dist;
double                  \rand_start_time;
```

Temporary Variables Block

```
double                  start_time;
Packet *                pk_ptr;

List *                  gdf_list_ptr;
```

```
char *              gdf_entry_string;
List *              gdf_entry_ptr;
int                 gdf_entry_size;
char *              gdf_element_string;
int                 i,j;

char *              temp_msg_name;
char *              temp_destination;
char *              temp_distribution;
```

INIT STATE

```
/*** Enter executive ***/

op_ima_obj_attr_get (op_id_self(), "Application Start Time",
&start_time);

/* initialize variables */
act_connect = 0;
seed = 430;
srand(seed);
```

DELAY STATE

```
/*** Enter executive ***/

//create delay to allow all servers to register before tcp connections
are established
op_intrpt_schedule_self (op_sim_time() + 1, 10000);
```

ESTCONN STATE

```
/*** Exit executive ***/

/* establish connection with TEU */
ici_ptr_teu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_teu);
op_ici_attr_set (ici_ptr_teu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_teu, "Service", "FDDI - source WSEU");
op_ici_attr_set (ici_ptr_teu, "Remote Port", 2);
op_ici_attr_set (ici_ptr_teu, "Local Port", 5);
op_ici_attr_set (ici_ptr_teu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_teu, "Remote Address", "TEU");
tpal_objid = op_topo_assoc (op_id_self(), OPC_TOPO_ASSOC_IN,
OPC_OBJMTYPE_MODULE, 0);
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with HEU */
ici_ptr_heu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_heu);
op_ici_attr_set (ici_ptr_heu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_heu, "Service", "FDDI - source WSEU");
op_ici_attr_set (ici_ptr_heu, "Remote Port", 2);
op_ici_attr_set (ici_ptr_heu, "Local Port", 6);
op_ici_attr_set (ici_ptr_heu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_heu, "Remote Address", "HEU");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with CCS */
ici_ptr_ccs = op_ici_create ("tpal_req");
```

```
op_ici_install (ici_ptr_ccs);
op_ici_attr_set (ici_ptr_ccs, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_ccs, "Service", "FDDI - source WSEU");
op_ici_attr_set (ici_ptr_ccs, "Remote Port", 2);
op_ici_attr_set (ici_ptr_ccs, "Local Port", 7);
op_ici_attr_set (ici_ptr_ccs, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_ccs, "Remote Address", "CCS");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);
```

WAIT_1 STATE

```
/*** Enter executive ***/

/* this state waits for all three tpal interfaces to be established */
```

CHK_ACT STATE

```
/*** Enter executive ***/

act_connect++;


/*** Exit executive ***/

/* proceed to send state only if all three tpal connections have been
opened */
/* otherwise, return to wait_1 state */
```

RD_GDF STATE

```
/*** Enter executive ***/

/* read the general data file (GDF) */
gdf_list_ptr = op_prg_gdf_read ("fddi_wseu_s1-9");

/* get the number of messages in the GDF file */
num_msg = op_prg_list_size (gdf_list_ptr);

/* create a structure and allocate mem for each message */
msg_to_send = (Msg_to_send *) op_prg_mem_alloc
(sizeof(Msg_to_send)*num_msg);

/* get each line in the GDF file */
for (i = 0; i < num_msg; i++)
      {
      /* get the ith line in the GDF file */
      gdf_entry_string = op_prg_list_access (gdf_list_ptr, i);

      /* decompose the string into its components */
      gdf_entry_ptr = op_prg_str_decomp (gdf_entry_string, ",");

      gdf_entry_size = op_prg_list_size (gdf_entry_ptr);        /* number
of fields in line */

      if (gdf_entry_size < 7)                          /* ensure valid # of
fields */
            {
            temp_msg_name = (char *) (op_prg_list_access (gdf_entry_ptr,
0));
            strcpy (msg_to_send[i].msg_name, temp_msg_name);
```

```
            temp_destination = (char *) (op_prg_list_access
(gdf_entry_ptr, 1));
            strcpy (msg_to_send[i].destination, temp_destination);
            msg_to_send[i].size_of_data = atoi (op_prg_list_access
(gdf_entry_ptr, 2));
            msg_to_send[i].frequency = atoi (op_prg_list_access
(gdf_entry_ptr, 3));
            msg_to_send[i].variance = atoi (op_prg_list_access
(gdf_entry_ptr, 4));
            temp_distribution = (char *) (op_prg_list_access
(gdf_entry_ptr, 5));
            strcpy (msg_to_send[i].distribution, temp_distribution);
            }
      }

/* deallocate the table list and contents */
op_prg_list_free (gdf_list_ptr);
op_prg_mem_free (gdf_list_ptr);

/* schedule interrupts for each message with op_intrpt_code = x*/
for (x = 0; x < num_msg; x++)
      {
      rand_start_time = ((rand() % 50) + op_dist_uniform(2));    /* start
all interrupts randomly within the first 50 sec */

      op_intrpt_schedule_self ((op_sim_time () + rand_start_time), x);
/* schedule the first message to be sent at the random start time */
      }
```

WAIT STATE

```
/*** Enter executive ***/

// this state does nothing except wait.
// when it is time for a message to be sent, the process proceeds to the
next state


/*** Exit executive ***/

intrpt_code = op_intrpt_code ();  /* store original op_intrpt_code */
```

SEND STATE

```
/*** Enter executive ***/

pk_ptr = op_pk_create_fmt ("AAAV-FDDI_pk");

/* set size of packet */
op_pk_total_size_set (pk_ptr, msg_to_send[intrpt_code].size_of_data);

//op_pk_nfd_set (pk_ptr, "PUIT", 0);       /* place holders for future
use */
//op_pk_nfd_set (pk_ptr, "Data", "test");

if (strcmp (msg_to_send[intrpt_code].destination, "HEU") == 0)
      {
      op_ici_install (ici_ptr_heu);
      }

else if (strcmp (msg_to_send[intrpt_code].destination, "CCS") == 0)
```

```
        {
        op_ici_install (ici_ptr_ccs);
        }

else if (strcmp (msg_to_send[intrpt_code].destination, "TEU") == 0)
        {
        op_ici_install (ici_ptr_teu);
        }

else
        {
        /* must be a type-o in the GDF */
        printf ("invalid destination in WSEU file on line %d\n",
intrpt_code);
        op_ici_install (ici_ptr_heu);  /* default to send to HEU */
        }

/* send the packet */
op_pk_send (pk_ptr, 0);
op_ici_install (OPC_NIL);


/*** Exit executive ***/

/* to make the message generator process truly random, we need to set the
next time this message is sent at another randomly generated time */

do
        {
        if (strcmp(msg_to_send[intrpt_code].distribution, "CONSTANT") == 0)
                {
                msg_to_send[intrpt_code].send_rate =
msg_to_send[intrpt_code].frequency;
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "UNIFORM")
== 0)
                {
                msg_to_send[intrpt_code].send_rate = op_dist_uniform
(msg_to_send[intrpt_code].frequency);
                }
        else if (strcmp(msg_to_send[intrpt_code].distribution, "NORMAL") ==
0)
                {
                gen_dist = op_dist_load ("normal",
msg_to_send[intrpt_code].frequency, msg_to_send[intrpt_code].variance);
                msg_to_send[intrpt_code].send_rate = op_dist_outcome
(gen_dist);
                }
} while (msg_to_send[intrpt_code].send_rate <= 0);   /* repeat until the
send rate is not equal to zero */

/* schedule another interrupt at this frequency */
op_intrpt_schedule_self (op_sim_time () + (double) 1.0/ ((double)
msg_to_send[intrpt_code].send_rate * 1/3600), intrpt_code);
```

APPENDIX E. OPNET CODE FOR MSG_RCVR MODULES

This appendix provides the OPNET source code for the *aaav_msg_rcvr* node modules. The code includes comments when appropriate.

The *aaav_msg_rcvr* state diagram is shown in Figure 12 and is repeated below in Figure 42.

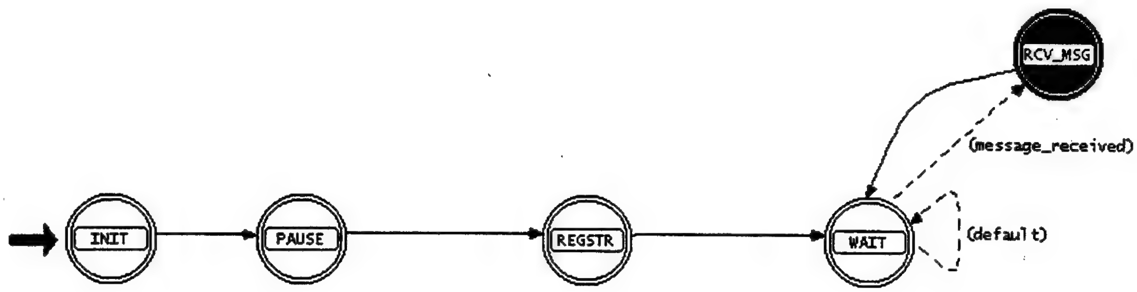

Figure 42. State Transition Diagram of the *aaav_msg_rcvr* Module.

OPNET CODE FOR AAAV_MSG_RCVR

Header Block
Deborah G. Peyton
May 1999

```
#include "tpal.h"
#define message_received (op_intrpt_type() == OPC_INTRPT_STRM)
```

State Variables Block

```
Objid        \tpal_objid;
Ici *        \ici_ptr;
int          \i;
```

Temporary Variables Block

```
Packet*      pk_ptr;
```

INIT STATE

```
/*** Enter executive ***/

op_intrpt_schedule_self (op_sim_time(), 0);
```

PAUSE STATE

```
/*** Enter executive ***/

op_intrpt_schedule_self (op_sim_time(), 0);
```

REGSTR STATE

```
/*** Enter executive ***/

op_intrpt_schedule_self (op_sim_time(), 0);


/*** Exit executive ***/

// get TPAL object id
tpal_objid = op_topo_assoc (op_id_self(), OPC_TOPO_ASSOC_IN,
OPC_OBJMTYPE_MODULE, 0);

for (i = 1; i <= 4; i++)
      {
      ici_ptr = op_ici_create ("tpal_serv_reg");

      // set up TCP connection
      op_ici_attr_set (ici_ptr, "Protocol", "tcp");
      op_ici_attr_set (ici_ptr, "Port", i);
      op_ici_attr_set (ici_ptr, "Service Name", "FDDI Application-TCP");
      op_ici_attr_set (ici_ptr, "Popularity", 1.0);
      op_ici_install (ici_ptr);
```

```
op_intrpt_force_remote (TPALC_CMD_SERV_REG, tpal_objid);

op_ici_destroy (ici_ptr);

ici_ptr = op_ici_create ("tpal_req");

// listen on TCP connection
op_ici_attr_set (ici_ptr, "flags", TPALC_OPT_PASSIVE);
op_ici_attr_set (ici_ptr, "Remote Address", TpalC_Host_Unspec);
op_ici_attr_set (ici_ptr, "Service", "FDDI Application - TCP");
op_ici_attr_set (ici_ptr, "Remote Port", TpalC_Port_Unspec);
op_ici_attr_set (ici_ptr, "Local Port", i);
op_ici_attr_set (ici_ptr, "Protocol", "tcp");
op_ici_install (ici_ptr);
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

op_ici_destroy (ici_ptr);
}
```

WAIT STATE

```
/*** Enter executive ***/

//this state waits for a message to be received
```

RCV_MSG STATE

```
/*** Enter executive ***/

/* get the incoming packet and its attributes */
pk_ptr = op_pk_get (op_intrpt_strm ());
op_pk_ici_get (pk_ptr);

/* packet processing would occur here */


/*** Exit executive ***/

/* destroy the packet */
op_pk_destroy (pk_ptr);
```

APPENDIX F. SCENARIO DEVELOPMENT SOURCE DOCUMENTATION

This appendix contains an excerpt from the High Speed Data Bus ICD provided by the AAAV Project Office. The ICD Appendix A [16] contains a Microsoft Excel spreadsheet identifying all High Speed Data Bus messages, signal name(s), source CSCI, and destination CSCI, while including information. The spreadsheets were modified to include estimates of message distributions and transmission frequencies as well as identification of the source bus (when necessary). These estimates were reviewed and accepted by the AAAV Project Office [18]. The scenarios used during the simulation of the Vetronics System High Speed Data Bus model were developed using the information contained in Appendix A of the High Speed Data Bus ICD [16] in addition to the message distribution and transmission frequency estimates. These Excel spreadsheets provide the necessary information to accurately simulate the High Speed Data Bus model.

Figure 43 shows the High Speed Data Bus interfaces for the AAAV-P Vetronics System. Table 34 shows the modified H-MPA_TO_NAVSA message set contained in Appendix A of the ICD. It includes the estimates of message distributions and transmission frequencies as well as identification of the source bus. The ICD contains a total of twenty-one Excel spreadsheets similar to the one shown in Table 34. Each Excel spreadsheet contained in the ICD Appendix A corresponds to an interface identified in Figure 43.

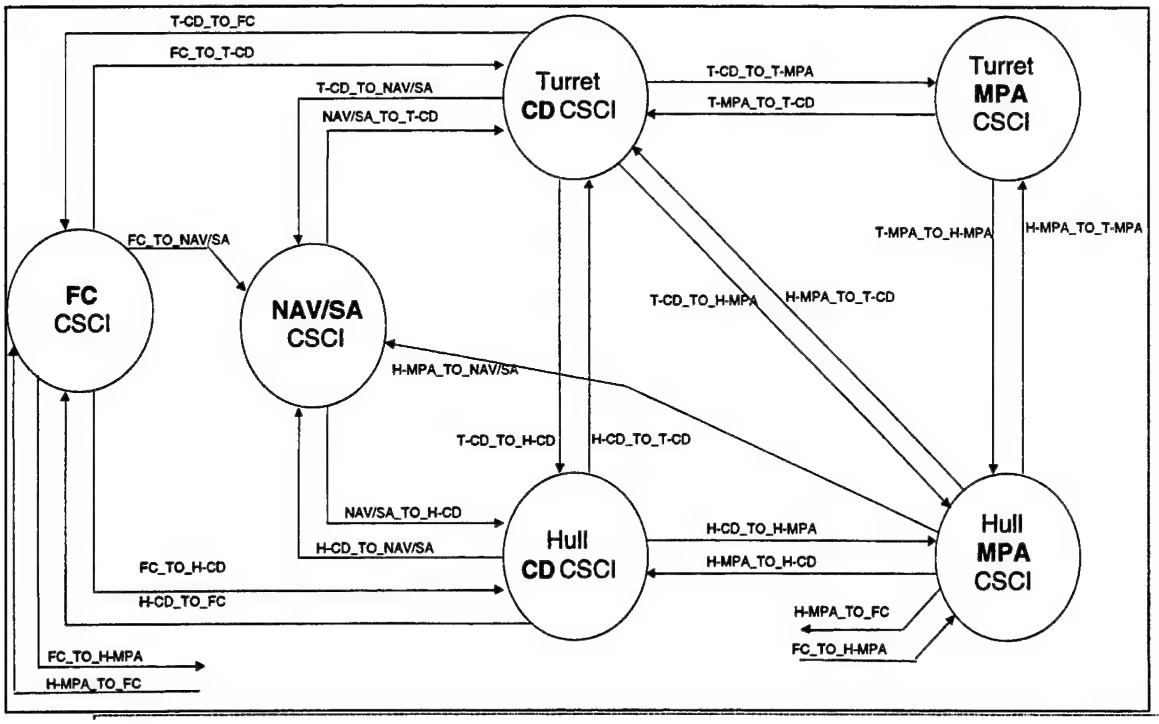

Figure 43. High Speed Data Bus Interface Diagram. [6].

RDD	Message	Signal Name	Source CSCI	Destination CSCI	Type/Unit	Source Bus	Distribution	Frequency (x times hr)	Rate	Enum Values	Range	Res	Notes	Data Provided B
ADT	TRANSMISSION_STATUS_MSG		MPA	CD	Enum	CB	Normal	60	Upon Change	FIRST_RANGE, SECOND_RANGE, THIRD_RANGE, FOURTH_RANGE, FIFTH_RANGE, SIXTH_RANGE, PIVOT_1, PIVOT_2, PIVOT_3, PIVOT_4, NEUTRAL, REVERSE_1, REVERSE_2, REVERSE_3, REVERSE_4				Harold
ADT	WATER_JET_STATUS_MSG		MPA	CD	Enum	UB	Normal	5	Upon Change	Water_Jets_Engaged, Water_Jets_Disengag ed				Harold
ADT/FM	ADT_TECU3_MSG		MPA	CD	Data Item	UB	Uniform	5	Upon Change				8 byte message	Harold
ADT/FM	ADT_TECUDM1_MSG		MPA	CD	Data Item	UB	Uniform	5	Upon Change				maximum 40 byte message	Harold
ADT/FM	SHIFT_INHIBIT_STATUS_MSG		MPA	CD	Enum	UB	Uniform	5	Upon Change	Shift_Inhibited, Shift_Not_Inhibited				Harold
AFES	AFES_READY_FOR_2ND_SHOT_MSG		MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	AFES System failure	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	AFES System is Off	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	APU 1 Sensor Fault	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	APU 2 Sensor Fault	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	APU Bottle Continuity Fault	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	APU Bottle Discharged	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	APU Bottle Fail to Flow Fault	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl
AFES/FM	AFES_MONITOR_MSG	APU Bottle Pressure Fault	MPA	CD	Condition	AFES 422	Normal	3	Upon Change					Gahl

Table 34. Excerpt of High Speed Data Bus ICD, Appendix A. [16].

APPENDIX G. HIGH SPEED DATA BUS MESSAGE TRAFFIC FILES

This appendix contains the complete set of general data files (GDFs) used during the simulation of the High Speed Data Bus. Table 18 identifies the network node, scenario, and associated GDF used during the simulation efforts, and is repeated in Table 35 for reference.

	Network Node			
Scenario	**TEU**	**WSEU**	**HEU**	**CCS**
1	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s1	fddi_ccs_s1-3
2	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s1-3
3	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s1-3
4	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s4-5
5	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s4-5
6	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s6-7-10
7	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s6-7-10
8	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s2-10-even	fddi_ccs_s8-9-11
9	fddi_teu_all	fddi_wseu_s1-9	fddi_heu_s3-11-odd	fddi_ccs_s8-9-11
10	fddi_teu_all	fddi_wseu_s10-11	fddi_heu_s2-10-even	fddi_ccs_s6-7-10
11	fddi_teu_all	fddi_wseu_s10-11	fddi_heu_s3-11-odd	fddi_ccs_s8-9-11

Table 35. Network Nodes, Scenarios, and Associated General Data File.

fddi_ccs_1-3

```
# CCS FDDI messages for Scenarios 1, 2, and 3
#
#message name,destination,data size(bits),frequency(times per hour),variability(=0 if N/A),distribution
#CCS-NAVSA to TEU-CD
Communication Alerts,TEU,32,120,2,NORMAL
X Window Display Items,TEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,TEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,TEU,32,3,1,NORMAL
 NAV_SA_CWA_FAULT_MSG,TEU,32,3,1,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_SOURCE,TEU,32,5,1,NORMAL
NO_DATA_SOURCE,TEU,32,5,1,NORMAL
#
#CCS-NAVSA to HEU-CD
Communication Alerts,HEU,32,120,2,NORMAL
X Window Display Items,HEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,HEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,HEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,HEU,32,3,1,NORMAL
NAV_SA_CWA_FAULT_MSG,HEU,32,3,1,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
```

```
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_SOURCE,HEU,32,5,1,NORMAL
NO_DATA_SOURCE,HEU,32,5,1,NORMAL
#
#end of message list for Scenarios 1, 2, and 3
```

fddi_ccs_4-5

```
# CCS FDDI messages for Scenarios 4 and 5
#
#message name,destination,data size(bits),frequency(times per hour),variability(=0 if N/A),distribution
#CCS-NAVSA to TEU-CD
Communication Alerts,TEU,32,120,2,NORMAL
X Window Display Items,TEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,TEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,TEU,32,3,1,NORMAL
 NAV_SA_CWA_FAULT_MSG,TEU,32,3,1,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_SOURCE,TEU,32,5,1,NORMAL
NO_DATA_SOURCE,TEU,32,5,1,NORMAL
#
#CCS-NAVSA to HEU-CD
Communication Alerts,HEU,32,120,2,NORMAL
X Window Display Items,HEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,HEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,HEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,HEU,32,3,1,NORMAL
NAV_SA_CWA_FAULT_MSG,HEU,32,3,1,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
```

```
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_SOURCE,HEU,32,5,1,NORMAL
NO_DATA_SOURCE,HEU,32,5,1,NORMAL
#
#CCS-NAVSA to TEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_TEU,TEU,4000,180,20,NORMAL
#
#CCS-NAVSA to HEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_HEU,HEU,4000,180,20,NORMAL
#
#CCS-NAVSA to WSEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_WSEU,WSEU,4000,180,20,NORMAL
#
#end of message list for Scenarios 4 and 5
```

fddi_ccs_6-7-10

```
# CCS FDDI messages for Scenarios 6, 7 and 10
#
#message name,destination,data size(bits),frequency(times per hour),variability(=0 if N/A),distribution
#CCS-NAVSA to TEU-CD
Communication Alerts,TEU,32,120,2,NORMAL
X Window Display Items,TEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,TEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,TEU,32,3,1,NORMAL
 NAV_SA_CWA_FAULT_MSG,TEU,32,3,1,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_SOURCE,TEU,32,5,1,NORMAL
NO_DATA_SOURCE,TEU,32,5,1,NORMAL
#
#CCS-NAVSA to HEU-CD
Communication Alerts,HEU,32,120,2,NORMAL
X Window Display Items,HEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,HEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,HEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,HEU,32,3,1,NORMAL
NAV_SA_CWA_FAULT_MSG,HEU,32,3,1,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
```

```
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_SOURCE,HEU,32,5,1,NORMAL
NO_DATA_SOURCE,HEU,32,5,1,NORMAL
#
#CCS-NAVSA to TEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_TEU,TEU,4000,180,20,NORMAL
#
#CCS-NAVSA to HEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_HEU,HEU,4000,180,20,NORMAL
#
#CCS-NAVSA to WSEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_WSEU,WSEU,4000,180,20,NORMAL
#
#CCS-NAVSA to TEU-CD
#Map Database Message Traffic - low load
MAP_DATA_TO_TEU,TEU,4000000,120,0,CONSTANT
#
#CCS-NAVSA to HEU-CD
#Map Database Message Traffic - low load
MAP_DATA_TO_HEU,HEU,4000000,120,0,CONSTANT
#
#CCS-NAVSA to WSEU-CD
#Map Database Message Traffic - low load
MAP_DATA_TO_WSEU,WSEU,4000000,120,0,CONSTANT
#
#end of message list for Scenarios 6, 7, and 10
```

fddi_ccs_8-9-11

CCS FDDI messages for Scenarios 8, 9, and 11
#
#message name,destination,data size(bits),frequency(times per hour),variability(if applicable),distribution
#CCS-NAVSA to TEU-CD
Communication Alerts,TEU,32,120,2,NORMAL
X Window Display Items,TEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,TEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,TEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,TEU,32,3,1,NORMAL
NAV_SA_CWA_FAULT_MSG,TEU,32,3,1,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
HEADING_DATA,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_MGRS,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_LAT_LONG,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_UTM,TEU,32,600,20,NORMAL
POSITION_SOURCE,TEU,32,5,1,NORMAL
NO_DATA_SOURCE,TEU,32,5,1,NORMAL
#
#CCS-NAVSA to HEU-CD
Communication Alerts,HEU,32,120,2,NORMAL
X Window Display Items,HEU,32,240,4,NORMAL
CCS_MMU_FAULT_MSG,HEU,32,3,1,NORMAL
NAV_SA_FMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PMODE_PREP_STATUS,HEU,32,3,1,NORMAL
NAV_SA_PWR_DWN_FAILED_MSG,HEU,32,3,1,NORMAL
NAV_SA_PWR_STATUS,HEU,32,3,1,NORMAL
NAV_SA_CWA_FAULT_MSG,HEU,32,3,1,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
HEADING_DATA,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL

POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_MGRS,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_LAT_LONG,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_UTM,HEU,32,600,20,NORMAL
POSITION_SOURCE,HEU,32,5,1,NORMAL
NO_DATA_SOURCE,HEU,32,5,1,NORMAL
#
#CCS-NAVSA to TEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_TEU,TEU,4000,180,20,NORMAL
#
#CCS-NAVSA to HEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_HEU,HEU,4000,180,20,NORMAL
#
#CCS-NAVSA to WSEU-CD
#Operational Message Traffic (OMT)
OP_MSG_TO_WSEU,WSEU,4000,180,20,NORMAL
#
#CCS-NAVSA to TEU-CD
#Map Database Message Traffic - high load
MAP_DATA_TO_TEU,TEU,8000000,120,0,CONSTANT
#
#CCS-NAVSA to HEU-CD
#Map Database Message Traffic - high load
MAP_DATA_TO_HEU,HEU,8000000,120,0,CONSTANT
#
#CCS-NAVSA to WSEU-CD
#Map Database Message Traffic - high load
MAP_DATA_TO_WSEU,WSEU,8000000,120,0,CONSTANT
#
#end of message list for Scenarios 8, 9, and 11

fddi_heu_s1

```
# HEU FDDI messages for Scenario 1
#
#HEU-CD to TEU-MPA
#message name,destination,data size(bits),frequency(times per hour),variability (=0 if N/A),distribution
DISCHARGE_2ND_SHOT_MSG,TEU,32,1,1,NORMAL
ALARM_ON_OFF_MSG,TEU,32,5,1,NORMAL
APU_START_MSG,TEU,32,5,1,NORMAL
APU_STOP_MSG,TEU,32,5,1,NORMAL
START_GLOW_PLUGS_MSG,TEU,32,5,1,NORMAL
AUTO_BILGE_BUTTON_MSG,TEU,32,5,1,NORMAL
BILGE_PRE_OPS,TEU,32,5,1,NORMAL
MANUAL_OP_ELEC_MSG,TEU,32,5,1,NORMAL
MANUAL_OP_ENG_MSG,TEU,32,5,1,NORMAL
MANUAL_OP_HYD_MSG,TEU,32,5,1,NORMAL
APU_PREHEAT_MSG,TEU,32,5,1,NORMAL
COOLING_CONTROL_MSG,TEU,32,5,1,NORMAL
ENG_PREHEAT_MSG,TEU,32,5,1,NORMAL
HEATER_CONTROL_MSG,TEU,32,5,1,NORMAL
TEMPERATURE_CONTROL_MSG,TEU,32,5,1,NORMAL
VENTILATION_CONTROL_MSG,TEU,32,5,1,NORMAL
EMER_ENGINE_SHUT_DOWN_MSG,TEU,32,5,1,NORMAL
ENGINE_BATTLE_SHORT_MSG,TEU,32,5,1,NORMAL
ENGINE_IMMED_START_MSG,TEU,32,5,1,NORMAL
ENGINE_START_MSG,TEU,32,5,1,NORMAL
ENGINE_STOP_MSG,TEU,32,5,1,NORMAL
THROTTLE_OVERRIDE_MSG,TEU,32,5,1,NORMAL
PORT_FUEL_VALVE_CLOSED_MSG,TEU,32,5,1,NORMAL
STBD_FUEL_VALVE_CLOSED_MSG,TEU,32,5,1,NORMAL
VEHICLE_LIGHTS_CONTROL_MSG,TEU,32,5,1,NORMAL
SEND_HEU_GPP2_PBIT_MSG,TEU,32,2,1,NORMAL
SEND_TEU_GPP2_PBIT_MSG,TEU,32,2,1,NORMAL
LOWER_RAMP_MSG,TEU,32,5,1,NORMAL
RAISE_RAMP_MSG,TEU,32,5,1,NORMAL
DEPLOY_MSG,TEU,32,5,1,NORMAL
DEPLOY_MSG,TEU,32,5,1,NORMAL
GUN_CLEAR_MSG,TEU,32,5,1,NORMAL
HSA_PRE_WATER_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_ABORT_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_OVERRIDE_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_RETRY_MSG,TEU,32,5,1,NORMAL
OVERRIDE_MSG,TEU,32,5,1,NORMAL
RETRACT_MSG,TEU,32,5,1,NORMAL
```

RETRACT_MSG,TEU,32,5,1,NORMAL
TRANS_FLAP_MSG,TEU,32,5,1,NORMAL
HYD_CROSSOVER_MSG,TEU,32,5,1,NORMAL
CHG_FMODE_PREP,TEU,32,5,1,NORMAL
CHG_PMODE_PREP,TEU,32,5,1,NORMAL
EXECUTE_FMODE_CHG,TEU,32,5,1,NORMAL
EXECUTE_PMODE_CHG,TEU,32,5,1,NORMAL
PWR_CHG_PREP,TEU,32,5,1,NORMAL
PWR_CHG_PREP,TEU,32,5,1,NORMAL
PWR_DOWN_FAIL_MSG,TEU,32,5,1,NORMAL
PWR_DWN_CMD,TEU,32,5,1,NORMAL
PWR_DWN_PREP,TEU,32,5,1,NORMAL
NAV_SHUTDOWN_REQUEST,TEU,32,5,1,NORMAL
NAV_STARTUP_REQUEST,TEU,32,5,1,NORMAL
NBC_AUTO_MSG,TEU,32,5,1,NORMAL
NBC_DET_WRN_MSG,TEU,32,5,1,NORMAL
NBC_POWER_MSG,TEU,32,5,1,NORMAL
SYS_RECONFIG_MSG,TEU,32,5,1,NORMAL
BALLAST_DOORS_MSG,TEU,32,5,1,NORMAL
HULL_SALVO_SELECT_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_ARM_SAFE_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_MAN_AUTO_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_SEMI_FULL_SELECT_FDDI,TEU,32,5,1,NORMAL
RUN_ROS_BIT_FDDI,TEU,32,5,1,NORMAL
VC_HULL_1_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_1_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_2_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_2_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_INVENTORY_SELECT_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_1_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_1_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_2_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_2_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
SILENT_WATCH_CONTROL_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_ABORT_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_RETRY_MSG,TEU,32,5,1,NORMAL
ADJUST_TRACK_TENSION_MSG,TEU,32,5,1,NORMAL
EXTEND_ITT_MSG,TEU,32,5,1,NORMAL
EXTEND_ITT_MSG,TEU,32,5,1,NORMAL
MANUAL_DEPLOY_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_DEPLOY_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL

PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
RETRACT_ITT_MSG,TEU,32,5,1,NORMAL
RETRACT_ITT_MSG,TEU,32,5,1,NORMAL
SUSP_CONTINUE_MSG,TEU,32,5,1,NORMAL
SUSP_OVERRIDE_MSG,TEU,32,5,1,NORMAL
SUSP_PRE_CANCEL_MSG,TEU,32,5,1,NORMAL
SUSP_PRE_WATER_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_ABORT_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_OVERRIDE_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_RETRY_MSG,TEU,32,5,1,NORMAL
APPENDAGE_CONTINUE_MSG,TEU,32,5,1,NORMAL
APPENDAGE_STOP_MSG,TEU,32,5,1,NORMAL
CB_CONTROL_MSG,TEU,32,5,1,NORMAL
#
#HEU-CD to TEU-CD
INITIATE MUTE SIGNAL,TEU,32,10,1,NORMAL
HEU_GPP1_CBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP1_PBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP1_SERIAL_ST_MSG,TEU,48,20,1,NORMAL
SEND_HEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
SEND_TEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
SFM_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_D_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_G_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_HSA_IND_MANAGE_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_HSA_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_QUESTION_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_ADJ_TRACK_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_IND_MANAGE__ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SYS_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_TC_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_VC_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_WARNING_CAUTION_MSG,TEU,32,20,1,NORMAL
SFM_ZEROIZE_ADVISORY_MSG,TEU,32,20,1,NORMAL
TEU_GPP1_CBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
TEU_GPP1_SERIAL_ST_MSG,TEU,48,20,1,NORMAL
CD_HEU_PMODE_PREP_STATUS,TEU,32,20,1,NORMAL
CHG_FMODE_PREP,TEU,32,3,1,NORMAL
CHG_PMODE_PREP,TEU,32,3,1,NORMAL
EXECUTE_FMODE_CHG,TEU,32,3,1,NORMAL
EXECUTE_PMODE_CHG,TEU,32,3,1,NORMAL

CD_HEU_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
CD_HEU_PWR_STATUS,TEU,32,3,1,NORMAL
LAND_MODE_MSG,TEU,32,3,1,NORMAL
PWR_CHG_PREP,TEU,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
PWR_DWN_PREP,TEU,32,3,1,NORMAL
SMM_IND_CTRL,TEU,32,20,1,NORMAL
CD_TEU_FMODE_PREP_STATUS,TEU,32,20,1,NORMAL
CD_TEU_PWR_STATUS,TEU,32,20,1,NORMAL
TRANSITION_MODE_MSG,TEU,32,20,1,NORMAL
WATER_MODE_MSG,TEU,32,20,1,NORMAL
#
#HEU-CD to CCS-NAVSA
X Window Key Press Events,CCS,32,240,5,NORMAL
CHG_FMODE_PREP,CCS,32,3,1,NORMAL
CHG_PMODE_PREP,CCS,32,3,1,NORMAL
EXECUTE_FMODE_CHG,CCS,32,3,1,NORMAL
EXECUTE_PMODE_CHG,CCS,32,3,1,NORMAL
PWR_CHG_PREP,CCS,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,CCS,32,3,1,NORMAL
PWR_DWN_PREP,CCS,32,3,1,NORMAL
Cursor Data,CCS,32,3600,20,NORMAL
Cursor Select,CCS,32,3600,20,NORMAL
HEADING_FORMAT_REQ_CDPD,CCS,32,240,5,NORMAL
HEADING_FORMAT_REQ_CDPT,CCS,32,240,5,NORMAL
HEADING_FORMAT_REQ_CDPV,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPD,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPT,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPV,CCS,32,240,5,NORMAL
Keypad_data,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPD,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPT,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPV,CCS,32,240,5,NORMAL
MOSB Selected,CCS,32,240,5,NORMAL
Position Navigation Data,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPD,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPT,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPV,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPD,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPT,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPV,CCS,32,240,5,NORMAL
#
#HEU-CD to WSEU-FC
AIR_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
AMMO_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
BARO_PRESSURE_CD_FDDI,WSEU,32,20,1,NORMAL

```
BATTLESIGHT_AP_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_COAX_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_HE_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
CD_AZ_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_EL_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_FUNCTION_FDDI,WSEU,32,5,1,NORMAL
CD_OPERATION_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
ENTRY_STATUS_FDDI,WSEU,32,5,1,NORMAL
HATCH_OVERRIDE_FDDI,WSEU,32,5,1,NORMAL
HOLD_BALLISTICS_RESET,WSEU,32,5,1,NORMAL
HTPS_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
MANUAL_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
NEW_AMMO_SUBDES,WSEU,32,5,1,NORMAL
PITCH_ROLL_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
RAISE_GUN_MSG_FDDI,WSEU,32,5,1,NORMAL
SUBDES_AMMO_CAN,WSEU,32,1,1,NORMAL
SUBDES_FOR_AMMO_CAN,WSEU,32,5,1,NORMAL
VC_ARM_SAFE_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_BATTLE_SHORT_REQ_FDDI,WSEU,32,1,1,NORMAL
VC_FIRE_RATE_SELECTION_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_PWR_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_STAB_SELECT_FDDI,WSEU,32,5,1,NORMAL
CHG_FMODE_PREP,WSEU,32,3,1,NORMAL
CHG_PMODE_PREP,WSEU,32,3,1,NORMAL
EXECUTE_FMODE_CHG,WSEU,32,3,1,NORMAL
EXECUTE_PMODE_CHG,WSEU,32,3,1,NORMAL
PWR_CHG_PREP,WSEU,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,WSEU,32,3,1,NORMAL
PWR_DWN_PREP,WSEU,32,3,1,NORMAL
#
#HEU-MPA to TEU-CD
#source buses: GPS422, AMB, NAV422, AFES422, ROS422, or other (not Utility or CAN
bus)
APPENDAGE_STOP_STATUS_MSG,TEU,32,3,1,NORMAL
GUN_MSG,TEU,32,3,1,NORMAL
COMM_FAULT_MSG,TEU,32,5,1,NORMAL
CRUISE_CONTROL_STATUS_MSG,TEU,32,720,5,NORMAL
ENG_BATTLE_SHORT_ACTIVE_MSG,TEU,32,3,1,NORMAL
THROTTLE_OVERRIDE_STATUS,TEU,32,5,1,NORMAL
HEU_GPP2_CBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP2_PBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP2_SERIAL_ST_MSG,TEU,32,3,1,NORMAL
```

TEU_GPP2_CBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP2_PBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP2_SERIAL_ST_MSG,TEU,32,3,1,NORMAL
INIT_POWERUP_MASTER,TEU,32,1,1,NORMAL
INIT_POWERUP_TURRET,TEU,32,1,1,NORMAL
MPA_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
MPA_PMODE_CHANGE_STATUS,TEU,32,5,1,NORMAL
MPA_PMODE_PREP_STATUS,TEU,32,5,1,NORMAL
MPA_PWR_STATUS,TEU,32,5,1,NORMAL
PWR_DOWN_FAILURE_MSG,TEU,32,1,1,NORMAL
PWR_STARTUP_MSG,TEU,32,3,1,NORMAL
SYS_RECONFIG_STATUS_MSG,TEU,32,5,1,NORMAL
FLASH_HULL_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
SILENT_WATCH_ON_MSG,TEU,32,3,1,NORMAL
SILENT_WATCH_ALLOWED_MSG,TEU,32,3,1,NORMAL
NO.4 PORT ROADARM STUCK,TEU,32,2,1,NORMAL
RECONFIG_SUSP_MSG,TEU,32,5,1,NORMAL
SUSP_MSG,TEU,32,5,1,NORMAL
AFES_READY_FOR_2ND_SHOT_MSG,TEU,32,3,1,NORMAL
AFES_MONITOR_MSG,TEU,32,3,1,NORMAL
AFES_START_MSG,TEU,32,3,1,NORMAL
RES_FLUID_LEVEL_MSG,TEU,32,5,1,NORMAL
HYD_START_MSG,TEU,32,5,1,NORMAL
BACKUP_TIME,TEU,32,60,5,NORMAL
NAV_HEADING,TEU,32,3600,10,NORMAL
NAV_POSITION,TEU,32,3600,10,NORMAL
NAV_VELOCITY,TEU,32,3600,10,NORMAL
NO_DATA_SOURCE,TEU,32,5,1,NORMAL
NAV_FAULT_MSG,TEU,32,3,1,NORMAL
FLASH_HULL_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
HULL_1_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
HULL_2_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
HULL_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
HULL_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
OVHD_1_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
OVHD_2_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
ROS_ARM_SAFE_STATUS_FDDI,TEU,32,5,1,NORMAL
ROS_C_W_A_MESSAGES_FDDI,TEU,32,3,1,NORMAL
ROS_MODE_STATUS_FDDI,TEU,32,3,1,NORMAL
TUBE1_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE10_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE11_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE12_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE13_STATUS_FDDI,TEU,32,30,2,NORMAL

TUBE14_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE15_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE16_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE17_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE18_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE19_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE2_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE20_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE21_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE22_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE23_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE24_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE25_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE26_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE27_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE28_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE29_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE3_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE30_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE31_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE32_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE4_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE5_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE6_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE7_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE8_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE9_STATUS_FDDI,TEU,32,30,2,NORMAL
ROS_FAULT_MSG,TEU,32,5,1,NORMAL
#
#HEU-MPA to WSEU-FC
#source bus: NAV422,NAV423,NAV424,NAV425 (not Utility or CAN bus)
BACKUP_TIME,WSEU,32,3600,0,CONSTANT
PITCH_ANGLE_FDDI,WSEU,32,720000,0,CONSTANT
ROLL_ANGLE_FDDI,WSEU,32,720000,0,CONSTANT
NAV_VELOCITY,WSEU,32,600,10,NORMAL
#
#HEU-MPA to CCS-NAVSA
#source bus: NAV422,NAV423,NAV424,NAV425 (not Utility or CAN bus)
BACKUP_TIME,CCS,32,60,2,NORMAL
NAV_HEADING,CCS,32,3600,10,NORMAL
NAV_POSITION,CCS,32,3600,10,NORMAL
NAV_VELOCITY,CCS,32,3600,10,NORMAL
NO_DATA_SOURCE,CCS,32,5,1,NORMAL
#
#end of message list for Scenario 1

fddi_heu_s2-10-even

```
# HEU FDDI messages for Scenarios 2, 4, 6, 8, and 10
#
#HEU-CD to TEU-MPA
#message name,destination,data size(bits),frequency(times per hour),variability (=0 if N/A),distribution
DISCHARGE_2ND_SHOT_MSG,TEU,32,1,1,NORMAL
ALARM_ON_OFF_MSG,TEU,32,5,1,NORMAL
APU_START_MSG,TEU,32,5,1,NORMAL
APU_STOP_MSG,TEU,32,5,1,NORMAL
START_GLOW_PLUGS_MSG,TEU,32,5,1,NORMAL
AUTO_BILGE_BUTTON_MSG,TEU,32,5,1,NORMAL
BILGE_PRE_OPS,TEU,32,5,1,NORMAL
MANUAL_OP_ELEC_MSG,TEU,32,5,1,NORMAL
MANUAL_OP_ENG_MSG,TEU,32,5,1,NORMAL
MANUAL_OP_HYD_MSG,TEU,32,5,1,NORMAL
APU_PREHEAT_MSG,TEU,32,5,1,NORMAL
COOLING_CONTROL_MSG,TEU,32,5,1,NORMAL
ENG_PREHEAT_MSG,TEU,32,5,1,NORMAL
HEATER_CONTROL_MSG,TEU,32,5,1,NORMAL
TEMPERATURE_CONTROL_MSG,TEU,32,5,1,NORMAL
VENTILATION_CONTROL_MSG,TEU,32,5,1,NORMAL
EMER_ENGINE_SHUT_DOWN_MSG,TEU,32,5,1,NORMAL
ENGINE_BATTLE_SHORT_MSG,TEU,32,5,1,NORMAL
ENGINE_IMMED_START_MSG,TEU,32,5,1,NORMAL
ENGINE_START_MSG,TEU,32,5,1,NORMAL
#ENGINE_STOP_MSG,TEU,32,5,1,NORMAL
THROTTLE_OVERRIDE_MSG,TEU,32,5,1,NORMAL
PORT_FUEL_VALVE_CLOSED_MSG,TEU,32,5,1,NORMAL
STBD_FUEL_VALVE_CLOSED_MSG,TEU,32,5,1,NORMAL
VEHICLE_LIGHTS_CONTROL_MSG,TEU,32,5,1,NORMAL
SEND_HEU_GPP2_PBIT_MSG,TEU,32,2,1,NORMAL
SEND_TEU_GPP2_PBIT_MSG,TEU,32,2,1,NORMAL
LOWER_RAMP_MSG,TEU,32,5,1,NORMAL
RAISE_RAMP_MSG,TEU,32,5,1,NORMAL
DEPLOY_MSG,TEU,32,5,1,NORMAL
DEPLOY_MSG,TEU,32,5,1,NORMAL
GUN_CLEAR_MSG,TEU,32,5,1,NORMAL
HSA_PRE_WATER_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_ABORT_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_OVERRIDE_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_RETRY_MSG,TEU,32,5,1,NORMAL
OVERRIDE_MSG,TEU,32,5,1,NORMAL
RETRACT_MSG,TEU,32,5,1,NORMAL
```

RETRACT_MSG,TEU,32,5,1,NORMAL
TRANS_FLAP_MSG,TEU,32,5,1,NORMAL
HYD_CROSSOVER_MSG,TEU,32,5,1,NORMAL
CHG_FMODE_PREP,TEU,32,5,1,NORMAL
CHG_PMODE_PREP,TEU,32,5,1,NORMAL
EXECUTE_FMODE_CHG,TEU,32,5,1,NORMAL
EXECUTE_PMODE_CHG,TEU,32,5,1,NORMAL
PWR_CHG_PREP,TEU,32,5,1,NORMAL
PWR_CHG_PREP,TEU,32,5,1,NORMAL
PWR_DOWN_FAIL_MSG,TEU,32,5,1,NORMAL
PWR_DWN_CMD,TEU,32,5,1,NORMAL
PWR_DWN_PREP,TEU,32,5,1,NORMAL
NAV_SHUTDOWN_REQUEST,TEU,32,5,1,NORMAL
NAV_STARTUP_REQUEST,TEU,32,5,1,NORMAL
NBC_AUTO_MSG,TEU,32,5,1,NORMAL
NBC_DET_WRN_MSG,TEU,32,5,1,NORMAL
NBC_POWER_MSG,TEU,32,5,1,NORMAL
SYS_RECONFIG_MSG,TEU,32,5,1,NORMAL
BALLAST_DOORS_MSG,TEU,32,5,1,NORMAL
HULL_SALVO_SELECT_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_ARM_SAFE_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_MAN_AUTO_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_SEMI_FULL_SELECT_FDDI,TEU,32,5,1,NORMAL
RUN_ROS_BIT_FDDI,TEU,32,5,1,NORMAL
VC_HULL_1_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_1_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_2_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_2_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_INVENTORY_SELECT_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_1_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_1_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_2_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_2_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
#SILENT_WATCH_CONTROL_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_ABORT_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_RETRY_MSG,TEU,32,5,1,NORMAL
ADJUST_TRACK_TENSION_MSG,TEU,32,5,1,NORMAL
EXTEND_ITT_MSG,TEU,32,5,1,NORMAL
EXTEND_ITT_MSG,TEU,32,5,1,NORMAL
MANUAL_DEPLOY_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_DEPLOY_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL

```
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
RETRACT_ITT_MSG,TEU,32,5,1,NORMAL
RETRACT_ITT_MSG,TEU,32,5,1,NORMAL
SUSP_CONTINUE_MSG,TEU,32,5,1,NORMAL
SUSP_OVERRIDE_MSG,TEU,32,5,1,NORMAL
SUSP_PRE_CANCEL_MSG,TEU,32,5,1,NORMAL
SUSP_PRE_WATER_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_ABORT_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_OVERRIDE_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_RETRY_MSG,TEU,32,5,1,NORMAL
APPENDAGE_CONTINUE_MSG,TEU,32,5,1,NORMAL
APPENDAGE_STOP_MSG,TEU,32,5,1,NORMAL
CB_CONTROL_MSG,TEU,32,5,1,NORMAL
#
#HEU-CD to TEU-CD
INITIATE MUTE SIGNAL,TEU,32,10,1,NORMAL
HEU_GPP1_CBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP1_PBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP1_SERIAL_ST_MSG,TEU,48,20,1,NORMAL
SEND_HEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
SEND_TEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
SFM_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_D_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_G_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_HSA_IND_MANAGE_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_HSA_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_QUESTION_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_ADJ_TRACK_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_IND_MANAGE__ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SYS_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_TC_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_VC_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_WARNING_CAUTION_MSG,TEU,32,20,1,NORMAL
SFM_ZEROIZE_ADVISORY_MSG,TEU,32,20,1,NORMAL
TEU_GPP1_CBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
TEU_GPP1_SERIAL_ST_MSG,TEU,48,20,1,NORMAL
CD_HEU_PMODE_PREP_STATUS,TEU,32,20,1,NORMAL
CHG_FMODE_PREP,TEU,32,3,1,NORMAL
CHG_PMODE_PREP,TEU,32,3,1,NORMAL
EXECUTE_FMODE_CHG,TEU,32,3,1,NORMAL
EXECUTE_PMODE_CHG,TEU,32,3,1,NORMAL
```

CD_HEU_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
CD_HEU_PWR_STATUS,TEU,32,3,1,NORMAL
LAND_MODE_MSG,TEU,32,3,1,NORMAL
PWR_CHG_PREP,TEU,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
PWR_DWN_PREP,TEU,32,3,1,NORMAL
SMM_IND_CTRL,TEU,32,20,1,NORMAL
CD_TEU_FMODE_PREP_STATUS,TEU,32,20,1,NORMAL
CD_TEU_PWR_STATUS,TEU,32,20,1,NORMAL
TRANSITION_MODE_MSG,TEU,32,20,1,NORMAL
WATER_MODE_MSG,TEU,32,20,1,NORMAL
#
#HEU-CD to CCS-NAVSA
X Window Key Press Events,CCS,32,240,5,NORMAL
CHG_FMODE_PREP,CCS,32,3,1,NORMAL
CHG_PMODE_PREP,CCS,32,3,1,NORMAL
EXECUTE_FMODE_CHG,CCS,32,3,1,NORMAL
EXECUTE_PMODE_CHG,CCS,32,3,1,NORMAL
PWR_CHG_PREP,CCS,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,CCS,32,3,1,NORMAL
PWR_DWN_PREP,CCS,32,3,1,NORMAL
Cursor Data,CCS,32,3600,20,NORMAL
Cursor Select,CCS,32,3600,20,NORMAL
HEADING_FORMAT_REQ_CDPD,CCS,32,240,5,NORMAL
HEADING_FORMAT_REQ_CDPT,CCS,32,240,5,NORMAL
HEADING_FORMAT_REQ_CDPV,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPD,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPT,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPV,CCS,32,240,5,NORMAL
Keypad data,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPD,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPT,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPV,CCS,32,240,5,NORMAL
MOSB Selected,CCS,32,240,5,NORMAL
Position Navigation Data,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPD,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPT,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPV,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPD,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPT,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPV,CCS,32,240,5,NORMAL
#
#HEU-CD to WSEU-FC
AIR_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
AMMO_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
BARO_PRESSURE_CD_FDDI,WSEU,32,20,1,NORMAL

BATTLESIGHT_AP_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_COAX_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_HE_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
CD_AZ_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_EL_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_FUNCTION_FDDI,WSEU,32,5,1,NORMAL
CD_OPERATION_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
ENTRY_STATUS_FDDI,WSEU,32,5,1,NORMAL
HATCH_OVERRIDE_FDDI,WSEU,32,5,1,NORMAL
HOLD_BALLISTICS_RESET,WSEU,32,5,1,NORMAL
HTPS_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
MANUAL_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
NEW_AMMO_SUBDES,WSEU,32,5,1,NORMAL
PITCH_ROLL_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
RAISE_GUN_MSG_FDDI,WSEU,32,5,1,NORMAL
SUBDES_AMMO_CAN,WSEU,32,1,1,NORMAL
SUBDES_FOR_AMMO_CAN,WSEU,32,5,1,NORMAL
VC_ARM_SAFE_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_BATTLE_SHORT_REQ_FDDI,WSEU,32,1,1,NORMAL
VC_FIRE_RATE_SELECTION_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_PWR_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_STAB_SELECT_FDDI,WSEU,32,5,1,NORMAL
CHG_FMODE_PREP,WSEU,32,3,1,NORMAL
CHG_PMODE_PREP,WSEU,32,3,1,NORMAL
EXECUTE_FMODE_CHG,WSEU,32,3,1,NORMAL
EXECUTE_PMODE_CHG,WSEU,32,3,1,NORMAL
PWR_CHG_PREP,WSEU,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,WSEU,32,3,1,NORMAL
PWR_DWN_PREP,WSEU,32,3,1,NORMAL
#
#HEU-MPA to TEU-CD
#source buses: GPS422, AMB, NAV422, AFES422, ROS422, or other (not Utility or CAN bus)
APPENDAGE_STOP_STATUS_MSG,TEU,32,3,1,NORMAL
GUN_MSG,TEU,32,3,1,NORMAL
COMM_FAULT_MSG,TEU,32,5,1,NORMAL
CRUISE_CONTROL_STATUS_MSG,TEU,32,720,5,NORMAL
ENG_BATTLE_SHORT_ACTIVE_MSG,TEU,32,3,1,NORMAL
THROTTLE_OVERRIDE_STATUS,TEU,32,5,1,NORMAL
HEU_GPP2_CBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP2_PBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP2_SERIAL_ST_MSG,TEU,32,3,1,NORMAL

TEU_GPP2_CBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP2_PBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP2_SERIAL_ST_MSG,TEU,32,3,1,NORMAL
INIT_POWERUP_MASTER,TEU,32,1,1,NORMAL
INIT_POWERUP_TURRET,TEU,32,1,1,NORMAL
MPA_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
MPA_PMODE_CHANGE_STATUS,TEU,32,5,1,NORMAL
MPA_PMODE_PREP_STATUS,TEU,32,5,1,NORMAL
MPA_PWR_STATUS,TEU,32,5,1,NORMAL
PWR_DOWN_FAILURE_MSG,TEU,32,1,1,NORMAL
PWR_STARTUP_MSG,TEU,32,3,1,NORMAL
SYS_RECONFIG_STATUS_MSG,TEU,32,5,1,NORMAL
FLASH_HULL_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
SILENT_WATCH_ON_MSG,TEU,32,3,1,NORMAL
SILENT_WATCH_ALLOWED_MSG,TEU,32,3,1,NORMAL
NO.4 PORT ROADARM STUCK,TEU,32,2,1,NORMAL
RECONFIG_SUSP_MSG,TEU,32,5,1,NORMAL
SUSP_MSG,TEU,32,5,1,NORMAL
AFES_READY_FOR_2ND_SHOT_MSG,TEU,32,3,1,NORMAL
AFES_MONITOR_MSG,TEU,32,3,1,NORMAL
AFES_START_MSG,TEU,32,3,1,NORMAL
RES_FLUID_LEVEL_MSG,TEU,32,5,1,NORMAL
HYD_START_MSG,TEU,32,5,1,NORMAL
BACKUP_TIME,TEU,32,60,5,NORMAL
NAV_HEADING,TEU,32,3600,10,NORMAL
NAV_POSITION,TEU,32,3600,10,NORMAL
NAV_VELOCITY,TEU,32,3600,10,NORMAL
NO_DATA_SOURCE,TEU,32,5,1,NORMAL
NAV_FAULT_MSG,TEU,32,3,1,NORMAL
FLASH_HULL_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
HULL_1_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
HULL_2_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
HULL_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
HULL_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
OVHD_1_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
OVHD_2_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
ROS_ARM_SAFE_STATUS_FDDI,TEU,32,5,1,NORMAL
ROS_C_W_A_MESSAGES_FDDI,TEU,32,3,1,NORMAL
ROS_MODE_STATUS_FDDI,TEU,32,3,1,NORMAL
TUBE1_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE10_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE11_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE12_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE13_STATUS_FDDI,TEU,32,30,2,NORMAL

TUBE14_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE15_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE16_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE17_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE18_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE19_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE2_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE20_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE21_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE22_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE23_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE24_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE25_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE26_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE27_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE28_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE29_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE3_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE30_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE31_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE32_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE4_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE5_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE6_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE7_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE8_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE9_STATUS_FDDI,TEU,32,30,2,NORMAL
ROS_FAULT_MSG,TEU,32,5,1,NORMAL
#
#HEU-MPA to WSEU-FC
#source bus: NAV422,NAV423,NAV424,NAV425 (not Utility or CAN bus)
BACKUP_TIME,WSEU,32,3600,0,CONSTANT
PITCH_ANGLE_FDDI,WSEU,32,720000,0,CONSTANT
ROLL_ANGLE_FDDI,WSEU,32,720000,0,CONSTANT
NAV_VELOCITY,WSEU,32,600,10,NORMAL
#
#HEU-MPA to CCS-NAVSA
#source bus: NAV422,NAV423,NAV424,NAV425 (not Utility or CAN bus)
BACKUP_TIME,CCS,32,60,2,NORMAL
NAV_HEADING,CCS,32,3600,10,NORMAL
NAV_POSITION,CCS,32,3600,10,NORMAL
NAV_VELOCITY,CCS,32,3600,10,NORMAL
NO_DATA_SOURCE,CCS,32,5,1,NORMAL
#
#HEU-MPA to TEU-CD
#source bus: CAN Bus

```
TRANSMISSION_STATUS_MSG,TEU,32,60,5,NORMAL
ENGINE_DATA_MSG,TEU,32,18000,50,NORMAL
E NGINE_CWA_FAULT_MSG,TEU,32,5,1,NORMAL
ENG_MALFUNCTION_MSG,TEU,32,18000,50,NORMAL
ENGINE_STATUS_MSG,TEU,32,5,1,NORMAL
#
#HEU-MPA to TEU-CD
#source bus: Utility Bus - low load - all messages sent one time per hour
WATER_JET_STATUS_MSG,TEU,32,1,1,NORMAL
ADT_TECU3_MSG,TEU,64,1,1,NORMAL
ADT_TECUDM1_MSG,TEU,320,1,1,NORMAL
SHIFT_INHIBIT_STATUS_MSG,TEU,32,1,1,NORMAL
CB_STATUS_MSG,TEU,32,1,1,NORMAL
GLOW_PLUGS_STATUS_MSG,TEU,32,1,1,NORMAL
APU_CWA_FAULT_MSG,TEU,32,1,1,NORMAL
APU_STATUS_MSG,TEU,32,1,1,NORMAL
AUTO_BILGE_STATUS_MSG,TEU,32,1,1,NORMAL
BILGE_PRE_OPS_DONE,TEU,32,1,1,NORMAL
ELEC_BILGE_STATUS_MSG,TEU,32,1,1,NORMAL
ENG_BILGE_STATUS_MSG,TEU,32,1,1,NORMAL
HYD_BILGE_STATUS_MSG,TEU,32,1,1,NORMAL
MANUAL_CONTROL_ONLY_MSG,TEU,32,1,1,NORMAL
BILGE_PUMP_RPM_FAIL_MSG,TEU,32,1,1,NORMAL
HIGH_WATER_MSG,TEU,32,1,1,NORMAL
ECS_STATUS_MSG,TEU,32,1,1,NORMAL
ECS_STATUS_MSG,TEU,32,1,1,NORMAL
ECS_FAULT_MSG,TEU,32,1,1,NORMAL
FUEL_CWA_FAULT_MSG,TEU,32,1,1,NORMAL
BRAKE_STATUS_MSG,TEU,32,1,1,NORMAL
LOW_SYSTEM_VOLTAGE_MSG,TEU,32,1,1,NORMAL
VEHICLE_LIGHTS_STATUS_MSG,TEU,32,1,1,NORMAL
SYSTEM_VOLTAGE_MSG,TEU,32,1,1,NORMAL
MPA_CWA_FAULT_MSG,TEU,32,1,1,NORMAL
CD_FAULT_MSG,TEU,32,1,1,NORMAL
UB_ST_MSG,TEU,32,1,1,NORMAL
HATCH_CWA_STATUS_MSG,TEU,32,1,1,NORMAL
HATCH_STATUS_MSG,TEU,32,1,1,NORMAL
HSA_POSITION_MSG,TEU,32,1,1,NORMAL
HSA_DEPLOY_MSG,TEU,32,1,1,NORMAL
HSA_IND_MANAGE_MSG,TEU,32,1,1,NORMAL
HSA_RETRACT_MSG,TEU,32,1,1,NORMAL
CROSSOVER_MSG,TEU,32,1,1,NORMAL
HYD_POWER_MSG,TEU,32,1,1,NORMAL
NBC_AUTO_STATUS_MSG,TEU,32,1,1,NORMAL
NBC_DET_WRN_STATUS_MSG,TEU,32,1,1,NORMAL
NBC_PWR_STATUS_MSG,TEU,32,1,1,NORMAL
```

NBC_MSG,TEU,32,1,1,NORMAL
BALLAST_DOORS_STATUS_MSG,TEU,32,1,1,NORMAL
SUSP_POSITION_MSG,TEU,32,1,1,NORMAL
ADJ_TRACK_SUSP_MSG,TEU,32,1,1,NORMAL
IND_MAN_SUSP_MSG,TEU,32,1,1,NORMAL
#
#HEU-MPA to WSEU-FC
#source bus: Utility Bus - low load - all messages sent one time per hour
BOW_FLAP_LEVEL_STATUS,WSEU,32,1,1,NORMAL
CHECK_GUN_REQ_FDDI,WSEU,32,1,1,NORMAL
COAX_ARM_RPC_STATUS,WSEU,32,1,1,NORMAL
DRIVER_HATCH_STATUS,WSEU,32,1,1,NORMAL
DTV_Height_1_Status,WSEU,32,1,1,NORMAL
DTV_Height_2_Status,WSEU,32,1,1,NORMAL
LEFT_CARGO_HATCH_STATUS,WSEU,32,1,1,NORMAL
RIGHT_CARGO_HATCH_STATUS,WSEU,32,1,1,NORMAL
TRANSOM_FLAP_LEVEL_STATUS,WSEU,32,1,1,NORMAL
TROOP_CMDR_HATCH_STATUS,WSEU,32,1,1,NORMAL
HATCH_CWA_STATUS_MSG,WSEU,32,1,1,NORMAL
HATCH_STATUS_MSG,WSEU,32,1,1,NORMAL
#
#end of message list for Scenarios 2, 4, 6, 8, and 10

fddi_heu_s3-11-odd

```
# HEU FDDI messages for Scenarios 3, 5, 7, 9, and 11
#
#HEU-CD to TEU-MPA
#message name,destination,data size(bits),frequency(times per hour),variability (=0 if N/A),distribution
DISCHARGE_2ND_SHOT_MSG,TEU,32,1,1,NORMAL
ALARM_ON_OFF_MSG,TEU,32,5,1,NORMAL
APU_START_MSG,TEU,32,5,1,NORMAL
APU_STOP_MSG,TEU,32,5,1,NORMAL
START_GLOW_PLUGS_MSG,TEU,32,5,1,NORMAL
AUTO_BILGE_BUTTON_MSG,TEU,32,5,1,NORMAL
BILGE_PRE_OPS,TEU,32,5,1,NORMAL
MANUAL_OP_ELEC_MSG,TEU,32,5,1,NORMAL
MANUAL_OP_ENG_MSG,TEU,32,5,1,NORMAL
MANUAL_OP_HYD_MSG,TEU,32,5,1,NORMAL
APU_PREHEAT_MSG,TEU,32,5,1,NORMAL
COOLING_CONTROL_MSG,TEU,32,5,1,NORMAL
ENG_PREHEAT_MSG,TEU,32,5,1,NORMAL
HEATER_CONTROL_MSG,TEU,32,5,1,NORMAL
TEMPERATURE_CONTROL_MSG,TEU,32,5,1,NORMAL
VENTILATION_CONTROL_MSG,TEU,32,5,1,NORMAL
EMER_ENGINE_SHUT_DOWN_MSG,TEU,32,5,1,NORMAL
ENGINE_BATTLE_SHORT_MSG,TEU,32,5,1,NORMAL
ENGINE_IMMED_START_MSG,TEU,32,5,1,NORMAL
ENGINE_START_MSG,TEU,32,5,1,NORMAL
ENGINE_STOP_MSG,TEU,32,5,1,NORMAL
THROTTLE_OVERRIDE_MSG,TEU,32,5,1,NORMAL
PORT_FUEL_VALVE_CLOSED_MSG,TEU,32,5,1,NORMAL
STBD_FUEL_VALVE_CLOSED_MSG,TEU,32,5,1,NORMAL
VEHICLE_LIGHTS_CONTROL_MSG,TEU,32,5,1,NORMAL
SEND_HEU_GPP2_PBIT_MSG,TEU,32,2,1,NORMAL
SEND_TEU_GPP2_PBIT_MSG,TEU,32,2,1,NORMAL
LOWER_RAMP_MSG,TEU,32,5,1,NORMAL
RAISE_RAMP_MSG,TEU,32,5,1,NORMAL
DEPLOY_MSG,TEU,32,5,1,NORMAL
DEPLOY_MSG,TEU,32,5,1,NORMAL
GUN_CLEAR_MSG,TEU,32,5,1,NORMAL
HSA_PRE_WATER_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_ABORT_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_OVERRIDE_MSG,TEU,32,5,1,NORMAL
HSA_RECONFIG_RETRY_MSG,TEU,32,5,1,NORMAL
OVERRIDE_MSG,TEU,32,5,1,NORMAL
RETRACT_MSG,TEU,32,5,1,NORMAL
```

RETRACT_MSG,TEU,32,5,1,NORMAL
TRANS_FLAP_MSG,TEU,32,5,1,NORMAL
HYD_CROSSOVER_MSG,TEU,32,5,1,NORMAL
CHG_FMODE_PREP,TEU,32,5,1,NORMAL
CHG_PMODE_PREP,TEU,32,5,1,NORMAL
EXECUTE_FMODE_CHG,TEU,32,5,1,NORMAL
EXECUTE_PMODE_CHG,TEU,32,5,1,NORMAL
PWR_CHG_PREP,TEU,32,5,1,NORMAL
PWR_CHG_PREP,TEU,32,5,1,NORMAL
PWR_DOWN_FAIL_MSG,TEU,32,5,1,NORMAL
PWR_DWN_CMD,TEU,32,5,1,NORMAL
PWR_DWN_PREP,TEU,32,5,1,NORMAL
NAV_SHUTDOWN_REQUEST,TEU,32,5,1,NORMAL
NAV_STARTUP_REQUEST,TEU,32,5,1,NORMAL
NBC_AUTO_MSG,TEU,32,5,1,NORMAL
NBC_DET_WRN_MSG,TEU,32,5,1,NORMAL
NBC_POWER_MSG,TEU,32,5,1,NORMAL
SYS_RECONFIG_MSG,TEU,32,5,1,NORMAL
BALLAST_DOORS_MSG,TEU,32,5,1,NORMAL
HULL_SALVO_SELECT_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_ARM_SAFE_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_MAN_AUTO_SELECT_FDDI,TEU,32,5,1,NORMAL
ROS_SEMI_FULL_SELECT_FDDI,TEU,32,5,1,NORMAL
RUN_ROS_BIT_FDDI,TEU,32,5,1,NORMAL
VC_HULL_1_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_1_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_2_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_HULL_2_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_INVENTORY_SELECT_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_1_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_1_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_2_OVERRIDE_GO_FDDI,TEU,32,5,1,NORMAL
VC_OVHD_2_OVERRIDE_NO_GO_FDDI,TEU,32,5,1,NORMAL
SILENT_WATCH_CONTROL_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_ABORT_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
ADJ_TRACK_RETRY_MSG,TEU,32,5,1,NORMAL
ADJUST_TRACK_TENSION_MSG,TEU,32,5,1,NORMAL
EXTEND_ITT_MSG,TEU,32,5,1,NORMAL
EXTEND_ITT_MSG,TEU,32,5,1,NORMAL
MANUAL_DEPLOY_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_DEPLOY_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
MANUAL_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL

```
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
PRESET_RETRACT_HSU_MSG,TEU,32,5,1,NORMAL
RETRACT_ITT_MSG,TEU,32,5,1,NORMAL
RETRACT_ITT_MSG,TEU,32,5,1,NORMAL
SUSP_CONTINUE_MSG,TEU,32,5,1,NORMAL
SUSP_OVERRIDE_MSG,TEU,32,5,1,NORMAL
SUSP_PRE_CANCEL_MSG,TEU,32,5,1,NORMAL
SUSP_PRE_WATER_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_ABORT_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_IM_APPNDG_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_OVERRIDE_MSG,TEU,32,5,1,NORMAL
SUSP_RECONFIG_RETRY_MSG,TEU,32,5,1,NORMAL
APPENDAGE_CONTINUE_MSG,TEU,32,5,1,NORMAL
APPENDAGE_STOP_MSG,TEU,32,5,1,NORMAL
CB_CONTROL_MSG,TEU,32,5,1,NORMAL
#
#HEU-CD to TEU-CD
INITIATE MUTE SIGNAL,TEU,32,10,1,NORMAL
HEU_GPP1_CBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP1_PBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP1_SERIAL_ST_MSG,TEU,48,20,1,NORMAL
SEND_HEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
SEND_TEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
SFM_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_D_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_G_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_HSA_IND_MANAGE_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_HSA_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_QUESTION_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_ADJ_TRACK_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_IND_MANAGE__ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SUS_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_SYS_RECONFIG_ADVISORY_MSG,TEU,32,20,1,NORMAL
SFM_TC_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_VC_CW_LIST_MSG,TEU,32,20,1,NORMAL
SFM_WARNING_CAUTION_MSG,TEU,32,20,1,NORMAL
SFM_ZEROIZE_ADVISORY_MSG,TEU,32,20,1,NORMAL
TEU_GPP1_CBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP1_PBIT_MSG,TEU,32,5,1,NORMAL
TEU_GPP1_SERIAL_ST_MSG,TEU,48,20,1,NORMAL
CD_HEU_PMODE_PREP_STATUS,TEU,32,20,1,NORMAL
CHG_FMODE_PREP,TEU,32,3,1,NORMAL
CHG_PMODE_PREP,TEU,32,3,1,NORMAL
EXECUTE_FMODE_CHG,TEU,32,3,1,NORMAL
EXECUTE_PMODE_CHG,TEU,32,3,1,NORMAL
```

CD_HEU_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
CD_HEU_PWR_STATUS,TEU,32,3,1,NORMAL
LAND_MODE_MSG,TEU,32,3,1,NORMAL
PWR_CHG_PREP,TEU,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
PWR_DWN_PREP,TEU,32,3,1,NORMAL
SMM_IND_CTRL,TEU,32,20,1,NORMAL
CD_TEU_FMODE_PREP_STATUS,TEU,32,20,1,NORMAL
CD_TEU_PWR_STATUS,TEU,32,20,1,NORMAL
TRANSITION_MODE_MSG,TEU,32,20,1,NORMAL
WATER_MODE_MSG,TEU,32,20,1,NORMAL
#
#HEU-CD to CCS-NAVSA
X Window Key Press Events,CCS,32,240,5,NORMAL
CHG_FMODE_PREP,CCS,32,3,1,NORMAL
CHG_PMODE_PREP,CCS,32,3,1,NORMAL
EXECUTE_FMODE_CHG,CCS,32,3,1,NORMAL
EXECUTE_PMODE_CHG,CCS,32,3,1,NORMAL
PWR_CHG_PREP,CCS,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,CCS,32,3,1,NORMAL
PWR_DWN_PREP,CCS,32,3,1,NORMAL
Cursor Data,CCS,32,3600,20,NORMAL
Cursor Select,CCS,32,3600,20,NORMAL
HEADING_FORMAT_REQ_CDPD,CCS,32,240,5,NORMAL
HEADING_FORMAT_REQ_CDPT,CCS,32,240,5,NORMAL
HEADING_FORMAT_REQ_CDPV,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPD,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPT,CCS,32,240,5,NORMAL
HEADING_ORIENTATION_REQUEST_CDPV,CCS,32,240,5,NORMAL
Keypad data,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPD,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPT,CCS,32,240,5,NORMAL
MGRS_PRECISION_REQUEST_CDPV,CCS,32,240,5,NORMAL
MOSB Selected,CCS,32,240,5,NORMAL
Position Navigation Data,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPD,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPT,CCS,32,240,5,NORMAL
POSITION_FORMAT_REQUEST_CDPV,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPD,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPT,CCS,32,240,5,NORMAL
VELOCITY_FORMAT_REQ_CDPV,CCS,32,240,5,NORMAL
#
#HEU-CD to WSEU-FC
AIR_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
AMMO_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
BARO_PRESSURE_CD_FDDI,WSEU,32,20,1,NORMAL

BATTLESIGHT_AP_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_COAX_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_HE_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
CD_AZ_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_EL_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_FUNCTION_FDDI,WSEU,32,5,1,NORMAL
CD_OPERATION_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
ENTRY_STATUS_FDDI,WSEU,32,5,1,NORMAL
HATCH_OVERRIDE_FDDI,WSEU,32,5,1,NORMAL
HOLD_BALLISTICS_RESET,WSEU,32,5,1,NORMAL
HTPS_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
MANUAL_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
NEW_AMMO_SUBDES,WSEU,32,5,1,NORMAL
PITCH_ROLL_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
RAISE_GUN_MSG_FDDI,WSEU,32,5,1,NORMAL
SUBDES_AMMO_CAN,WSEU,32,1,1,NORMAL
SUBDES_FOR_AMMO_CAN,WSEU,32,5,1,NORMAL
VC_ARM_SAFE_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_BATTLE_SHORT_REQ_FDDI,WSEU,32,1,1,NORMAL
VC_FIRE_RATE_SELECTION_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_PWR_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_STAB_SELECT_FDDI,WSEU,32,5,1,NORMAL
CHG_FMODE_PREP,WSEU,32,3,1,NORMAL
CHG_PMODE_PREP,WSEU,32,3,1,NORMAL
EXECUTE_FMODE_CHG,WSEU,32,3,1,NORMAL
EXECUTE_PMODE_CHG,WSEU,32,3,1,NORMAL
PWR_CHG_PREP,WSEU,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,WSEU,32,3,1,NORMAL
PWR_DWN_PREP,WSEU,32,3,1,NORMAL
#
#HEU-MPA to TEU-CD
#source buses: GPS422, AMB, NAV422, AFES422, ROS422, or other (not Utility or CAN bus)
APPENDAGE_STOP_STATUS_MSG,TEU,32,3,1,NORMAL
GUN_MSG,TEU,32,3,1,NORMAL
COMM_FAULT_MSG,TEU,32,5,1,NORMAL
CRUISE_CONTROL_STATUS_MSG,TEU,32,720,5,NORMAL
ENG_BATTLE_SHORT_ACTIVE_MSG,TEU,32,3,1,NORMAL
THROTTLE_OVERRIDE_STATUS,TEU,32,5,1,NORMAL
HEU_GPP2_CBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP2_PBIT_MSG,TEU,32,20,1,NORMAL
HEU_GPP2_SERIAL_ST_MSG,TEU,32,3,1,NORMAL

TEU_GPP2_CBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP2_PBIT_MSG,TEU,32,20,1,NORMAL
TEU_GPP2_SERIAL_ST_MSG,TEU,32,3,1,NORMAL
INIT_POWERUP_MASTER,TEU,32,1,1,NORMAL
INIT_POWERUP_TURRET,TEU,32,1,1,NORMAL
MPA_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
MPA_PMODE_CHANGE_STATUS,TEU,32,5,1,NORMAL
MPA_PMODE_PREP_STATUS,TEU,32,5,1,NORMAL
MPA_PWR_STATUS,TEU,32,5,1,NORMAL
PWR_DOWN_FAILURE_MSG,TEU,32,1,1,NORMAL
PWR_STARTUP_MSG,TEU,32,3,1,NORMAL
SYS_RECONFIG_STATUS_MSG,TEU,32,5,1,NORMAL
FLASH_HULL_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
SILENT_WATCH_ON_MSG,TEU,32,3,1,NORMAL
SILENT_WATCH_ALLOWED_MSG,TEU,32,3,1,NORMAL
NO.4 PORT ROADARM STUCK,TEU,32,2,1,NORMAL
RECONFIG_SUSP_MSG,TEU,32,5,1,NORMAL
SUSP_MSG,TEU,32,5,1,NORMAL
AFES_READY_FOR_2ND_SHOT_MSG,TEU,32,3,1,NORMAL
AFES_MONITOR_MSG,TEU,32,3,1,NORMAL
AFES_START_MSG,TEU,32,3,1,NORMAL
RES_FLUID_LEVEL_MSG,TEU,32,5,1,NORMAL
HYD_START_MSG,TEU,32,5,1,NORMAL
BACKUP_TIME,TEU,32,60,5,NORMAL
NAV_HEADING,TEU,32,3600,10,NORMAL
NAV_POSITION,TEU,32,3600,10,NORMAL
NAV_VELOCITY,TEU,32,3600,10,NORMAL
NO_DATA_SOURCE,TEU,32,5,1,NORMAL
NAV_FAULT_MSG,TEU,32,3,1,NORMAL
FLASH_HULL_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
HULL_1_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
HULL_2_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
HULL_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
HULL_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_1_STATUS_FDDI,TEU,32,5,1,NORMAL
OVERHEAD_SALVO_2_STATUS_FDDI,TEU,32,5,1,NORMAL
OVHD_1_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
OVHD_2_NO_GO_COUNT_FDDI,TEU,32,5,1,NORMAL
ROS_ARM_SAFE_STATUS_FDDI,TEU,32,5,1,NORMAL
ROS_C_W_A_MESSAGES_FDDI,TEU,32,3,1,NORMAL
ROS_MODE_STATUS_FDDI,TEU,32,3,1,NORMAL
TUBE1_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE10_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE11_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE12_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE13_STATUS_FDDI,TEU,32,30,2,NORMAL

TUBE14_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE15_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE16_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE17_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE18_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE19_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE2_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE20_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE21_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE22_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE23_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE24_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE25_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE26_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE27_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE28_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE29_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE3_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE30_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE31_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE32_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE4_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE5_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE6_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE7_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE8_STATUS_FDDI,TEU,32,30,2,NORMAL
TUBE9_STATUS_FDDI,TEU,32,30,2,NORMAL
ROS_FAULT_MSG,TEU,32,5,1,NORMAL
#
#HEU-MPA to WSEU-FC
#source bus: NAV422,NAV423,NAV424,NAV425 (not Utility or CAN bus)
BACKUP_TIME,WSEU,32,3600,0,CONSTANT
PITCH_ANGLE_FDDI,WSEU,32,720000,0,CONSTANT
ROLL_ANGLE_FDDI,WSEU,32,720000,0,CONSTANT
NAV_VELOCITY,WSEU,32,600,10,NORMAL
#
#HEU-MPA to CCS-NAVSA
#source bus: NAV422,NAV423,NAV424,NAV425 (not Utility or CAN bus)
BACKUP_TIME,CCS,32,60,2,NORMAL
NAV_HEADING,CCS,32,3600,10,NORMAL
NAV_POSITION,CCS,32,3600,10,NORMAL
NAV_VELOCITY,CCS,32,3600,10,NORMAL
NO_DATA_SOURCE,CCS,32,5,1,NORMAL
#
#HEU-MPA to TEU-CD
#source bus: CAN Bus

TRANSMISSION_STATUS_MSG,TEU,32,60,5,NORMAL
ENGINE_DATA_MSG,TEU,32,18000,50,NORMAL
E NGINE_CWA_FAULT_MSG,TEU,32,5,1,NORMAL
ENG_MALFUNCTION_MSG,TEU,32,18000,50,NORMAL
ENGINE_STATUS_MSG,TEU,32,5,1,NORMAL
#
#HEU-MPA to TEU-CD
#source bus: Utility Bus - high load
WATER_JET_STATUS_MSG,TEU,32,5,1,NORMAL
ADT_TECU3_MSG,TEU,64,5,1,NORMAL
ADT_TECUDM1_MSG,TEU,320,5,1,NORMAL
SHIFT_INHIBIT_STATUS_MSG,TEU,32,5,1,NORMAL
CB_STATUS_MSG,TEU,32,60,5,NORMAL
GLOW_PLUGS_STATUS_MSG,TEU,32,3,1,NORMAL
APU_CWA_FAULT_MSG,TEU,32,3,1,NORMAL
APU_STATUS_MSG,TEU,32,3,1,NORMAL
AUTO_BILGE_STATUS_MSG,TEU,32,3,1,NORMAL
BILGE_PRE_OPS_DONE,TEU,32,3,1,NORMAL
ELEC_BILGE_STATUS_MSG,TEU,32,3,1,NORMAL
ENG_BILGE_STATUS_MSG,TEU,32,3,1,NORMAL
HYD_BILGE_STATUS_MSG,TEU,32,3,1,NORMAL
MANUAL_CONTROL_ONLY_MSG,TEU,32,3,1,NORMAL
BILGE_PUMP_RPM_FAIL_MSG,TEU,32,3,1,NORMAL
HIGH_WATER_MSG,TEU,32,3,1,NORMAL
ECS_STATUS_MSG,TEU,32,30,2,NORMAL
ECS_STATUS_MSG,TEU,32,30,2,NORMAL
ECS_FAULT_MSG,TEU,32,3,1,NORMAL
FUEL_CWA_FAULT_MSG,TEU,32,5,1,NORMAL
BRAKE_STATUS_MSG,TEU,32,5,1,NORMAL
LOW_SYSTEM_VOLTAGE_MSG,TEU,32,1,1,NORMAL
VEHICLE_LIGHTS_STATUS_MSG,TEU,32,5,1,NORMAL
SYSTEM_VOLTAGE_MSG,TEU,32,5,1,NORMAL
MPA_CWA_FAULT_MSG,TEU,32,3,1,NORMAL
CD_FAULT_MSG,TEU,32,3,1,NORMAL
UB_ST_MSG,TEU,32,5,1,NORMAL
HATCH_CWA_STATUS_MSG,TEU,32,30,2,NORMAL
HATCH_STATUS_MSG,TEU,32,30,2,NORMAL
HSA_POSITION_MSG,TEU,32,36000,50,NORMAL
HSA_DEPLOY_MSG,TEU,32,5,1,NORMAL
HSA_IND_MANAGE_MSG,TEU,32,5,1,NORMAL
HSA_RETRACT_MSG,TEU,32,5,1,NORMAL
CROSSOVER_MSG,TEU,32,2,1,NORMAL
HYD_POWER_MSG,TEU,32,5,1,NORMAL
NBC_AUTO_STATUS_MSG,TEU,32,1,1,NORMAL
NBC_DET_WRN_STATUS_MSG,TEU,32,1,1,NORMAL
NBC_PWR_STATUS_MSG,TEU,32,1,1,NORMAL

```
NBC_MSG,TEU,32,3,1,NORMAL
BALLAST_DOORS_STATUS_MSG,TEU,32,3,1,NORMAL
SUSP_POSITION_MSG,TEU,32,36000,50,NORMAL
ADJ_TRACK_SUSP_MSG,TEU,32,5,1,NORMAL
IND_MAN_SUSP_MSG,TEU,32,5,1,NORMAL
#
#HEU-MPA to WSEU-FC
#source bus: Utility Bus - high load
BOW_FLAP_LEVEL_STATUS,WSEU,32,600,20,NORMAL
CHECK_GUN_REQ_FDDI,WSEU,32,600,20,NORMAL
COAX_ARM_RPC_STATUS,WSEU,32,600,20,NORMAL
DRIVER_HATCH_STATUS,WSEU,32,360,10,NORMAL
DTV_Height_1_Status,WSEU,32,360,10,NORMAL
DTV_Height_2_Status,WSEU,32,360,10,NORMAL
LEFT_CARGO_HATCH_STATUS,WSEU,32,360,10,NORMAL
RIGHT_CARGO_HATCH_STATUS,WSEU,32,360,10,NORMAL
TRANSOM_FLAP_LEVEL_STATUS,WSEU,32,360,10,NORMAL
TROOP_CMDR_HATCH_STATUS,WSEU,32,360,10,NORMAL
HATCH_CWA_STATUS_MSG,WSEU,32,360,10,NORMAL
HATCH_STATUS_MSG,WSEU,32,36000,50,NORMAL
#
#end of message list for Scenarios 3, 5, 7, 9, and 11
```

fddi_teu_all

```
# TEU FDDI messages for all Scenarios
#
#TEU-CD to WSEU-FC
#message name,destination,data size(bits),frequency(times per hour),variability (=0 if N/A),distribution
AIR_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
AMMO_TEMP_CD_FDDI,WSEU,32,20,1,NORMAL
BARO_PRESSURE_CD_FDDI,WSEU,32,20,1,NORMAL
BATTLESIGHT_AP_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_COAX_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
BATTLESIGHT_HE_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
CD_AZ_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_EL_DATA_FDDI,WSEU,32,5,1,NORMAL
CD_FUNCTION_FDDI,WSEU,32,5,1,NORMAL
CD_OPERATION_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
CROSSWIND_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
ENTRY_STATUS_FDDI,WSEU,32,5,1,NORMAL
HATCH_OVERRIDE_FDDI,WSEU,32,5,1,NORMAL
HOLD_BALLISTICS_RESET,WSEU,32,5,1,NORMAL
HTPS_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
LEAD_VALUE_CD_FDDI,WSEU,32,5,1,NORMAL
MANUAL_RANGE_CD_FDDI,WSEU,32,5,1,NORMAL
NEW_AMMO_SUBDES,WSEU,32,5,1,NORMAL
PITCH_ROLL_MODE_CD_FDDI,WSEU,32,5,1,NORMAL
RAISE_GUN_MSG_FDDI,WSEU,32,5,1,NORMAL
SUBDES_AMMO_CAN,WSEU,32,1,1,NORMAL
SUBDES_FOR_AMMO_CAN,WSEU,32,5,1,NORMAL
VC_ARM_SAFE_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_BATTLE_SHORT_REQ_FDDI,WSEU,32,1,1,NORMAL
VC_FIRE_RATE_SELECTION_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_PWR_SELECT_FDDI,WSEU,32,5,1,NORMAL
VC_GTD_STAB_SELECT_FDDI,WSEU,32,5,1,NORMAL
CHG_FMODE_PREP,WSEU,32,3,1,NORMAL
CHG_PMODE_PREP,WSEU,32,3,1,NORMAL
EXECUTE_FMODE_CHG,WSEU,32,3,1,NORMAL
EXECUTE_PMODE_CHG,WSEU,32,3,1,NORMAL
PWR_CHG_PREP,WSEU,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,WSEU,32,3,1,NORMAL
PWR_DWN_PREP,WSEU,32,3,1,NORMAL
#
#TEU-CD to CCS-NAVSA
X Window Key Press Events,CCS,32,240,5,NORMAL
```

CHG_FMODE_PREP,CCS,32,3,1,NORMAL
CHG_PMODE_PREP,CCS,32,3,1,NORMAL
EXECUTE_FMODE_CHG,CCS,32,3,1,NORMAL
EXECUTE_PMODE_CHG,CCS,32,3,1,NORMAL
PWR_CHG_PREP,CCS,32,3,1,NORMAL
PWR_DWN_FAILED_MSG,CCS,32,3,1,NORMAL
PWR_DWN_PREP,CCS,32,3,1,NORMAL
#
#TEU-MPA to CCS-NAVSA
#source bus: NAV422,NAV423,NAV424,NAV425 (not Utility or CAN bus)
BACKUP_TIME,CCS,32,60,3,NORMAL
NAV_HEADING,CCS,32,3600,20,NORMAL
NAV_POSITION,CCS,32,3600,20,NORMAL
NAV_VELOCITY,CCS,32,3600,20,NORMAL
#
#end of message list for all Scenarios

fddi_wseu_s1-9

WSEU FDDI messages for Scenarios 1 through 9
#
#WSEU-FC to TEU-CD
#message name,destination,data size(bits),frequency(times per hour),variability (=0 if N/A),distribution
ADJUSTED_TURRET_POSITION_FDDI,TEU,32,36000,0,CONSTANT
AIR_TEMP_FDDI,TEU,32,60,5,NORMAL
AMMO_TEMP_FDDI,TEU,32,60,5,NORMAL
AMMO_TYPE_STATUS_FDDI,TEU,32,60,5,NORMAL
ARM_SAFE_STATUS_FDDI,TEU,32,60,5,NORMAL
BARO_PRESSURE_FDDI,TEU,32,60,5,NORMAL
BATTLESIGHT_AP_RANGE_FDDI,TEU,32,720,5,NORMAL
BATTLESIGHT_COAX_RANGE_FDDI,TEU,32,720,5,NORMAL
BATTLESIGHT_HE_RANGE_FDDI,TEU,32,720,5,NORMAL
CMS_AZ_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
CMS_BORESIGHT_COAX_AZ_FDDI,TEU,32,720,5,NORMAL
CMS_BORESIGHT_COAX_EL_FDDI,TEU,32,720,5,NORMAL
CMS_BORESIGHT_MAIN_AZ_FDDI,TEU,32,720,5,NORMAL
CMS_BORESIGHT_MAIN_EL_FDDI,TEU,32,720,5,NORMAL
CMS_EL_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
CROSSWIND_MODE_FDDI,TEU,32,60,5,NORMAL
CROSSWIND_VALUE_FDDI,TEU,32,60,5,NORMAL
CURRENT_AP_CAN_SUBDES_FDDI,TEU,32,60,5,NORMAL
CURRENT_COAX_CAN_SUBDES_FDDI,TEU,32,60,5,NORMAL
CURRENT_HE_CAN_SUBDES_FDDI,TEU,32,60,5,NORMAL
CURRENT_WPN_FDDI,TEU,32,60,5,NORMAL
CURRENT_WPN_STATUS_FDDI,TEU,32,60,5,NORMAL
FC_AZ_DATA_FDDI,TEU,32,60,5,NORMAL
FC_BATTLE_SHORT_STATUS_FDDI,TEU,32,1,1,NORMAL
FC_CONFIGURATION_FDDI,TEU,32,60,5,NORMAL
FC_EL_DATA_FDDI,TEU,32,60,5,NORMAL
FC_FUNCTION_FDDI,TEU,32,60,5,NORMAL
FC_OPERATION_FDDI,TEU,32,60,5,NORMAL
FIRE_RATE_STATUS_FDDI,TEU,32,60,5,NORMAL
GTD_AZ_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
GTD_EL_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
GTD_MODE_STATUS_FDDI,TEU,32,60,5,NORMAL
HELD_BALLISTICS_FC_FDDI,TEU,32,60,5,NORMAL
HELD_BALLISTICS_MSG,TEU,32,60,5,NORMAL
HTPS_MODE_FC_FDDI,TEU,32,60,5,NORMAL
LEAD_MODE_FDDI,TEU,32,60,5,NORMAL
LEAD_VALUE_FDDI,TEU,32,60,5,NORMAL
LOW_AMMO_STATUS_FDDI,TEU,32,60,5,NORMAL
MANUAL_RANGE_FDDI,TEU,32,60,5,NORMAL

OPEN_HATCH_INHIBIT_STATUS_FDDI,TEU,32,60,5,NORMAL
PITCH_ANGLE_FC_FDDI,TEU,32,36000,0,CONSTANT
PITCH_ROLL_MODE_FDDI,TEU,32,3600,20,NORMAL
PLUMB_SYNC_AZ_FDDI,TEU,32,3600,20,NORMAL
PLUMB_SYNC_EL_FDDI,TEU,32,3600,20,NORMAL
ROLL_ANGLE_FC_FDDI,TEU,32,36000,0,CONSTANT
ROUND_ZERO_AP_CAN_AZ_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_AP_CAN_EL_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_AZ_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_EL_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_AZ_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_EL_FDDI,TEU,32,30,2,NORMAL
FC_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
FC_PMODE_PREP_STATUS,TEU,32,3,1,NORMAL
FC_PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
FC_PWR_STATUS,TEU,32,3,1,NORMAL
#
#WSEU-FC to HEU-CD
ADJUSTED_TURRET_POSITION_FDDI,HEU,32,36000,0,CONSTANT
AIR_TEMP_FDDI,HEU,32,60,5,NORMAL
AMMO_TEMP_FDDI,HEU,32,60,5,NORMAL
AMMO_TYPE_STATUS_FDDI,HEU,32,60,5,NORMAL
ARM_SAFE_STATUS_FDDI,HEU,32,60,5,NORMAL
BARO_PRESSURE_FDDI,HEU,32,60,5,NORMAL
CMS_AZ_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
CMS_EL_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
CROSSWIND_MODE_FDDI,HEU,32,60,5,NORMAL
CROSSWIND_VALUE_FDDI,HEU,32,60,5,NORMAL
CURRENT_AP_CAN_SUBDES_FDDI,HEU,32,60,5,NORMAL
CURRENT_COAX_CAN_SUBDES_FDDI,HEU,32,60,5,NORMAL
CURRENT_HE_CAN_SUBDES_FDDI,HEU,32,60,5,NORMAL
CURRENT_WPN_FDDI,HEU,32,60,5,NORMAL
CURRENT_WPN_STATUS_FDDI,HEU,32,60,5,NORMAL
FC_AZ_DATA_FDDI,HEU,32,60,5,NORMAL
FC_BATTLE_SHORT_STATUS_FDDI,HEU,32,1,1,NORMAL
FC_CONFIGURATION_FDDI,HEU,32,60,5,NORMAL
FC_EL_DATA_FDDI,HEU,32,60,5,NORMAL
FC_FUNCTION_FDDI,HEU,32,60,5,NORMAL
FC_OPERATION_FDDI,HEU,32,60,5,NORMAL
FIRE_RATE_STATUS_FDDI,HEU,32,60,5,NORMAL
GTD_AZ_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
GTD_EL_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
GTD_MODE_STATUS_FDDI,HEU,32,60,5,NORMAL
HELD_BALLISTICS_FC_FDDI,HEU,32,60,5,NORMAL
HELD_BALLISTICS_MSG,HEU,32,60,5,NORMAL
HTPS_MODE_FC_FDDI,HEU,32,60,5,NORMAL

LEAD_MODE_FDDI,HEU,32,60,5,NORMAL
LEAD_VALUE_FDDI,HEU,32,60,5,NORMAL
LOW_AMMO_STATUS_FDDI,HEU,32,60,5,NORMAL
MANUAL_RANGE_FDDI,HEU,32,60,5,NORMAL
OPEN_HATCH_INHIBIT_STATUS_FDDI,HEU,32,60,5,NORMAL
PITCH_ANGLE_FC_FDDI,HEU,32,36000,0,CONSTANT
PITCH_ROLL_MODE_FDDI,HEU,32,3600,20,NORMAL
PLUMB_SYNC_AZ_FDDI,HEU,32,3600,20,NORMAL
PLUMB_SYNC_EL_FDDI,HEU,32,3600,20,NORMAL
ROLL_ANGLE_FC_FDDI,HEU,32,36000,0,CONSTANT
ROUND_ZERO_AP_CAN_AZ_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_AP_CAN_EL_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_AZ_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_EL_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_AZ_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_EL_FDDI,HEU,32,30,2,NORMAL
FC_FMODE_PREP_STATUS,HEU,32,3,1,NORMAL
FC_PMODE_PREP_STATUS,HEU,32,3,1,NORMAL
FC_PWR_DWN_FAILED_MSG,HEU,32,3,1,NORMAL
FC_PWR_STATUS,HEU,32,3,1,NORMAL
#
#WSEU-FC to HEU-MPA
GUN_POSITION_STATUS_FDDI,HEU,32,5,1,NORMAL
ARM_SAFE_STATUS_FDDI,HEU,32,360,5,NORMAL
CURRENT_WPN_FDDI,HEU,32,360,5,NORMAL
GUN_AZ_POSITION_FDDI,HEU,32,60,2,NORMAL
GUN_EL_POSITION_FDDI,HEU,32,60,2,NORMAL
#
#WSEU-FC to CCS-NAVSA
GUN_POSITION_EL_FDDI,CCS,32,3600,20,NORMAL
NAV_RANGE_FDDI,CCS,32,3600,20,NORMAL
TURRET_POSITION_NAV_FDDI,CCS,32,3600,20,NORMAL
#
#end of message list for Scenarios 1 through 9

fddi_wseu_s10-11

```
# WSEU FDDI messages for Scenarios 10 and 11
#
#WSEU-FC to TEU-CD
#message name,destination,data size(bits),frequency(times per hour),variability (=0 if N/A),distribution (NORMAL/CONSTANT/UNIFORM)
ADJUSTED_TURRET_POSITION_FDDI,TEU,32,36000,0,CONSTANT
AIR_TEMP_FDDI,TEU,32,60,5,NORMAL
AMMO_TEMP_FDDI,TEU,32,60,5,NORMAL
AMMO_TYPE_STATUS_FDDI,TEU,32,60,5,NORMAL
ARM_SAFE_STATUS_FDDI,TEU,32,60,5,NORMAL
BARO_PRESSURE_FDDI,TEU,32,60,5,NORMAL
BATTLESIGHT_AP_RANGE_FDDI,TEU,32,720,5,NORMAL
BATTLESIGHT_COAX_RANGE_FDDI,TEU,32,720,5,NORMAL
BATTLESIGHT_HE_RANGE_FDDI,TEU,32,720,5,NORMAL
CMS_AZ_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
CMS_BORESIGHT_COAX_AZ_FDDI,TEU,32,720,5,NORMAL
CMS_BORESIGHT_COAX_EL_FDDI,TEU,32,720,5,NORMAL
CMS_BORESIGHT_MAIN_AZ_FDDI,TEU,32,720,5,NORMAL
CMS_BORESIGHT_MAIN_EL_FDDI,TEU,32,720,5,NORMAL
CMS_EL_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
CROSSWIND_MODE_FDDI,TEU,32,60,5,NORMAL
CROSSWIND_VALUE_FDDI,TEU,32,60,5,NORMAL
CURRENT_AP_CAN_SUBDES_FDDI,TEU,32,60,5,NORMAL
CURRENT_COAX_CAN_SUBDES_FDDI,TEU,32,60,5,NORMAL
CURRENT_HE_CAN_SUBDES_FDDI,TEU,32,60,5,NORMAL
CURRENT_WPN_FDDI,TEU,32,60,5,NORMAL
CURRENT_WPN_STATUS_FDDI,TEU,32,60,5,NORMAL
FC_AZ_DATA_FDDI,TEU,32,60,5,NORMAL
FC_BATTLE_SHORT_STATUS_FDDI,TEU,32,1,1,NORMAL
FC_CONFIGURATION_FDDI,TEU,32,60,5,NORMAL
FC_EL_DATA_FDDI,TEU,32,60,5,NORMAL
FC_FUNCTION_FDDI,TEU,32,60,5,NORMAL
FC_OPERATION_FDDI,TEU,32,60,5,NORMAL
FIRE_RATE_STATUS_FDDI,TEU,32,60,5,NORMAL
GTD_AZ_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
GTD_EL_DRIFT_VALUE_FDDI,TEU,32,60,5,NORMAL
GTD_MODE_STATUS_FDDI,TEU,32,60,5,NORMAL
HELD_BALLISTICS_FC_FDDI,TEU,32,60,5,NORMAL
HELD_BALLISTICS_MSG,TEU,32,60,5,NORMAL
HTPS_MODE_FC_FDDI,TEU,32,60,5,NORMAL
LEAD_MODE_FDDI,TEU,32,60,5,NORMAL
LEAD_VALUE_FDDI,TEU,32,60,5,NORMAL
LOW_AMMO_STATUS_FDDI,TEU,32,60,5,NORMAL
MANUAL_RANGE_FDDI,TEU,32,60,5,NORMAL
```

OPEN_HATCH_INHIBIT_STATUS_FDDI,TEU,32,60,5,NORMAL
PITCH_ANGLE_FC_FDDI,TEU,32,36000,0,CONSTANT
PITCH_ROLL_MODE_FDDI,TEU,32,3600,20,NORMAL
PLUMB_SYNC_AZ_FDDI,TEU,32,3600,20,NORMAL
PLUMB_SYNC_EL_FDDI,TEU,32,3600,20,NORMAL
ROLL_ANGLE_FC_FDDI,TEU,32,36000,0,CONSTANT
ROUND_ZERO_AP_CAN_AZ_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_AP_CAN_EL_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_AZ_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_EL_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_AZ_FDDI,TEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_EL_FDDI,TEU,32,30,2,NORMAL
FC_FMODE_PREP_STATUS,TEU,32,3,1,NORMAL
FC_PMODE_PREP_STATUS,TEU,32,3,1,NORMAL
FC_PWR_DWN_FAILED_MSG,TEU,32,3,1,NORMAL
FC_PWR_STATUS,TEU,32,3,1,NORMAL
#
#WSEU-FC to HEU-CD
ADJUSTED_TURRET_POSITION_FDDI,HEU,32,36000,0,CONSTANT
AIR_TEMP_FDDI,HEU,32,60,5,NORMAL
AMMO_TEMP_FDDI,HEU,32,60,5,NORMAL
AMMO_TYPE_STATUS_FDDI,HEU,32,60,5,NORMAL
ARM_SAFE_STATUS_FDDI,HEU,32,60,5,NORMAL
BARO_PRESSURE_FDDI,HEU,32,60,5,NORMAL
CMS_AZ_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
CMS_EL_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
CROSSWIND_MODE_FDDI,HEU,32,60,5,NORMAL
CROSSWIND_VALUE_FDDI,HEU,32,60,5,NORMAL
CURRENT_AP_CAN_SUBDES_FDDI,HEU,32,60,5,NORMAL
CURRENT_COAX_CAN_SUBDES_FDDI,HEU,32,60,5,NORMAL
CURRENT_HE_CAN_SUBDES_FDDI,HEU,32,60,5,NORMAL
CURRENT_WPN_FDDI,HEU,32,60,5,NORMAL
CURRENT_WPN_STATUS_FDDI,HEU,32,60,5,NORMAL
FC_AZ_DATA_FDDI,HEU,32,60,5,NORMAL
FC_BATTLE_SHORT_STATUS_FDDI,HEU,32,1,1,NORMAL
FC_CONFIGURATION_FDDI,HEU,32,60,5,NORMAL
FC_EL_DATA_FDDI,HEU,32,60,5,NORMAL
FC_FUNCTION_FDDI,HEU,32,60,5,NORMAL
FC_OPERATION_FDDI,HEU,32,60,5,NORMAL
FIRE_RATE_STATUS_FDDI,HEU,32,60,5,NORMAL
GTD_AZ_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
GTD_EL_DRIFT_VALUE_FDDI,HEU,32,60,5,NORMAL
GTD_MODE_STATUS_FDDI,HEU,32,60,5,NORMAL
HELD_BALLISTICS_FC_FDDI,HEU,32,60,5,NORMAL
HELD_BALLISTICS_MSG,HEU,32,60,5,NORMAL
HTPS_MODE_FC_FDDI,HEU,32,60,5,NORMAL

```
LEAD_MODE_FDDI,HEU,32,60,5,NORMAL
LEAD_VALUE_FDDI,HEU,32,60,5,NORMAL
LOW_AMMO_STATUS_FDDI,HEU,32,60,5,NORMAL
MANUAL_RANGE_FDDI,HEU,32,60,5,NORMAL
OPEN_HATCH_INHIBIT_STATUS_FDDI,HEU,32,60,5,NORMAL
PITCH_ANGLE_FC_FDDI,HEU,32,36000,0,CONSTANT
PITCH_ROLL_MODE_FDDI,HEU,32,3600,20,NORMAL
PLUMB_SYNC_AZ_FDDI,HEU,32,3600,20,NORMAL
PLUMB_SYNC_EL_FDDI,HEU,32,3600,20,NORMAL
ROLL_ANGLE_FC_FDDI,HEU,32,36000,0,CONSTANT
ROUND_ZERO_AP_CAN_AZ_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_AP_CAN_EL_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_AZ_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_COAX_CAN_EL_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_AZ_FDDI,HEU,32,30,2,NORMAL
ROUND_ZERO_HE_CAN_EL_FDDI,HEU,32,30,2,NORMAL
FC_FMODE_PREP_STATUS,HEU,32,3,1,NORMAL
FC_PMODE_PREP_STATUS,HEU,32,3,1,NORMAL
FC_PWR_DWN_FAILED_MSG,HEU,32,3,1,NORMAL
FC_PWR_STATUS,HEU,32,3,1,NORMAL
#
#WSEU-FC to HEU-MPA
GUN_POSITION_STATUS_FDDI,HEU,32,5,1,NORMAL
ARM_SAFE_STATUS_FDDI,HEU,32,360,5,NORMAL
CURRENT_WPN_FDDI,HEU,32,360,5,NORMAL
GUN_AZ_POSITION_FDDI,HEU,32,60,2,NORMAL
GUN_EL_POSITION_FDDI,HEU,32,60,2,NORMAL
#
#WSEU-FC to CCS-NAVSA
GUN_POSITION_EL_FDDI,CCS,32,3600,20,NORMAL
NAV_RANGE_FDDI,CCS,32,3600,20,NORMAL
TURRET_POSITION_NAV_FDDI,CCS,32,3600,20,NORMAL
#
#WSEU-FC to TEU-CD
#BATTLESIGHT AND BORESIGHT related messages
BATTLESIGHT_AP_RANGE_FDDI,HEU,32,720,5,NORMAL
BATTLESIGHT_COAX_RANGE_FDDI,HEU,32,720,5,NORMAL
BATTLESIGHT_HE_RANGE_FDDI,HEU,32,720,5,NORMAL
CMS_BORESIGHT_COAX_AZ_FDDI,HEU,32,720,5,NORMAL
CMS_BORESIGHT_COAX_EL_FDDI,HEU,32,720,5,NORMAL
CMS_BORESIGHT_MAIN_AZ_FDDI,HEU,32,720,5,NORMAL
CMS_BORESIGHT_MAIN_EL_FDDI,HEU,32,720,5,NORMAL
#
#end of message list for Scenarios 10 and 11
```

APPENDIX H. OPNET CODE FOR STATISTICS COLLECTION

This appendix provides the modified OPNET source code for the heu_msg_gen and aaav_msg_rcvr node modules. The original code was modified in order to generate and collect user-defined statistics.

The SEND state of the heu_msg_gen process model was modified to include message information with each message that is transmitted by the HEU. The INIT and RCV_MSG states of the aaav_msg_rcvr process model were modified to generate and collect two statistics that measure the arrival rate of specific messages. The code includes comments when appropriate.

OPNET CODE FOR HEU_MSG_GEN

Header Block
Deborah G. Peyton
May 1999

```
#include "tpal.h"
#include <string.h>
#include <stdlib.h>

#define NEEDCONN (op_intrpt_type() == OPC_INTRPT_SELF)
#define active ((op_intrpt_type() == OPC_INTRPT_REMOTE) &&
(op_intrpt_code() == TPALC_EV_CONF_OPEN))
#define not_all_act (act_connect <= 2)
#define all_act (act_connect == 3)
#define time_to_send (op_intrpt_type() == OPC_INTRPT_SELF)

/* define the structure for the message parameters */
typedef struct
      {
      char msg_name[30];
      char destination[10];
      int size_of_data;
      int frequency;
      int variance;
      char distribution[15];
      double send_rate;
      } Msg_to_send;
```

State Variables Block

```
Objid              \tpal_objid;
Ici *              \ici_ptr_wseu;
Ici *              \ici_ptr_teu;
Ici *              \ici_ptr_ccs;

/* the number of messages in the GDF */
int                \num_msg;

Msg_to_send *      \msg_to_send;
int                \intrpt_code;

/* counter */
int                \x;

/* the number of active TCP connections established */
int                \act_connect;

unsigned           \seed;
Distribution *     \gen_dist;
double             \rand_start_time;
int                \z;
```

Temporary Variables Block

```
double             start_time;
Packet *           pk_ptr;
```

```
List *              gdf_list_ptr;
char *              gdf_entry_string;
List *              gdf_entry_ptr;
int                 gdf_entry_size;
char *              gdf_element_string;
int                 i,j;

char *              temp_msg_name;
char *              temp_destination;
char *              temp_distribution;
```

INIT STATE

```
/*** Enter executive ***/

op_ima_obj_attr_get (op_id_self(), "Application Start Time",
&start_time);

/* initialize variables */
act_connect = 0;
seed = 648;
srand(seed);
```

DELAY STATE

```
/*** Enter executive ***/

/* create delay to allow all servers to register before tcp connections
are established */
op_intrpt_schedule_self (op_sim_time() + 1, 10000);
```

ESTCONN STATE

```
/*** Exit executive ***/

/* establish connection with WSEU */
ici_ptr_wseu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_wseu);
op_ici_attr_set (ici_ptr_wseu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_wseu, "Service", "FDDI - source HEU");
op_ici_attr_set (ici_ptr_wseu, "Remote Port", 3);
op_ici_attr_set (ici_ptr_wseu, "Local Port", 5);
op_ici_attr_set (ici_ptr_wseu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_wseu, "Remote Address", "WSEU");
tpal_objid = op_topo_assoc (op_id_self(), OPC_TOPO_ASSOC_IN,
OPC_OBJMTYPE_MODULE, 0);
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with TEU */
ici_ptr_teu = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_teu);
op_ici_attr_set (ici_ptr_teu, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_teu, "Service", "FDDI - source HEU");
op_ici_attr_set (ici_ptr_teu, "Remote Port", 3);
op_ici_attr_set (ici_ptr_teu, "Local Port", 6);
op_ici_attr_set (ici_ptr_teu, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_teu, "Remote Address", "TEU");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);

/* establish connection with CCS */
```

```
ici_ptr_ccs = op_ici_create ("tpal_req");
op_ici_install (ici_ptr_ccs);
op_ici_attr_set (ici_ptr_ccs, "flags", TPALC_OPT_ACTIVE);
op_ici_attr_set (ici_ptr_ccs, "Service", "FDDI - source HEU");
op_ici_attr_set (ici_ptr_ccs, "Remote Port", 3);
op_ici_attr_set (ici_ptr_ccs, "Local Port", 7);
op_ici_attr_set (ici_ptr_ccs, "Protocol", "tcp");
op_ici_attr_set (ici_ptr_ccs, "Remote Address", "CCS");
op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);
```

WAIT_1 STATE

```
/*** Enter executive ***/

/* this state waits for all three tpal interfaces to be established */
```

CHK_ACT STATE

```
/*** Enter executive ***/

act_connect++;


/*** Exit executive ***/

/* proceed to SEND state only if all three tpal connections have been
opened */
/* otherwise, return to WAIT_1 state */
```

RD_GDF STATE

```
/*** Enter executive ***/

/* read the general data file (GDF) */
gdf_list_ptr = op_prg_gdf_read ("fddi_heu_s1");

/* get the number of messages in the GDF file */
num_msg = op_prg_list_size (gdf_list_ptr);

/* create a structure and allocate mem for each message */
msg_to_send = (Msg_to_send *) op_prg_mem_alloc
(sizeof(Msg_to_send)*num_msg);

/* get each line in the GDF file */
for (i = 0; i < num_msg; i++)
      {
      /* get the ith line in the GDF file */
      gdf_entry_string = op_prg_list_access (gdf_list_ptr, i);

      /* decompose the string into its components */
      gdf_entry_ptr = op_prg_str_decomp (gdf_entry_string, ",");

      /* get the number of fields in the line */
      gdf_entry_size = op_prg_list_size (gdf_entry_ptr);

      if (gdf_entry_size < 7)     /* ensure valid # of fields */
            {
            /* break out the fields into structure */
            temp_msg_name = (char *) (op_prg_list_access (gdf_entry_ptr,
0));
```

```
            strcpy (msg_to_send[i].msg_name, temp_msg_name);
            temp_destination = (char *) (op_prg_list_access
(gdf_entry_ptr, 1));
            strcpy (msg_to_send[i].destination, temp_destination);
            msg_to_send[i].size_of_data = atoi (op_prg_list_access
(gdf_entry_ptr, 2));
            msg_to_send[i].frequency = atoi (op_prg_list_access
(gdf_entry_ptr, 3));
            msg_to_send[i].variance = atoi (op_prg_list_access
(gdf_entry_ptr, 4));
            temp_distribution = (char *) (op_prg_list_access
(gdf_entry_ptr, 5));
            strcpy (msg_to_send[i].distribution, temp_distribution);
            }
      }

/* deallocate the table list and contents */
op_prg_list_free (gdf_list_ptr);
op_prg_mem_free (gdf_list_ptr);

/* schedule interrupts for each message with op_intrpt_code = x*/
for (x = 0; x < num_msg; x++)
      {
      /* start all interrupts randomly within the first 50 sec */
      rand_start_time = ((rand() % 50) + op_dist_uniform(2));

      /* schedule first message to be sent at the random start time */
      op_intrpt_schedule_self ((op_sim_time () + rand_start_time), x);
      }
```

WAIT STATE

```
/*** Enter executive ***/

/* this state does nothing except wait. */
/* when it is time for a message to be sent, the process proceeds to the
next state */


/*** Exit executive ***/

intrpt_code = op_intrpt_code ();   /* store original op_intrpt_code */
```

SEND STATE

```
/*** Enter executive ***/

pk_ptr = op_pk_create_fmt ("AAAV-FDDI_pk");

op_pk_total_size_set (pk_ptr, msg_to_send[intrpt_code].size_of_data);
/* set size of packet */

/* put a counter in the PUIT field - for debugging purposes only -
counter has no meaning */
op_pk_nfd_set (pk_ptr, "PUIT", z);
z++;

/* make a copy of the memory containing the message to send */
copy_msg_to_send = op_prg_mem_copy_create (&msg_to_send[intrpt_code],
sizeof(Msg_to_send));
```

```
/* put the copied message information into the data field of the message
*/
op_pk_nfd_set (pk_ptr, "Data", copy_msg_to_send, op_prg_mem_copy_create,
op_prg_mem_free, sizeof(Msg_to_send));

if (strcmp (msg_to_send[intrpt_code].destination, "TEU") == 0)
	{
	op_ici_install (ici_ptr_teu);
	}

else if (strcmp (msg_to_send[intrpt_code].destination, "CCS") == 0)
	{
	op_ici_install (ici_ptr_ccs);
	}

else if (strcmp (msg_to_send[intrpt_code].destination, "WSEU") == 0)
	{
	op_ici_install (ici_ptr_wseu);
	}

else
	{
	/* must be a type-o in the GDF */
	printf ("invalid destination in HEU file on line %d\n",
intrpt_code);
	op_ici_install (ici_ptr_teu);		/* default to send to TEU */
	}

/* send the packet */
op_pk_send (pk_ptr, 0);
op_ici_install (OPC_NIL);


/*** Exit executive ***/

/* to make the message generator process truly random, we need to set the
next time this message is sent at another randomly generated time */

do
	{
	if (strcmp(msg_to_send[intrpt_code].distribution, "CONSTANT") == 0)
		{
		msg_to_send[intrpt_code].send_rate =
msg_to_send[intrpt_code].frequency;
		}
	else if (strcmp(msg_to_send[intrpt_code].distribution, "UNIFORM")
== 0)
		{
		msg_to_send[intrpt_code].send_rate = op_dist_uniform
(msg_to_send[intrpt_code].frequency);
		}
	else if (strcmp(msg_to_send[intrpt_code].distribution, "NORMAL") ==
0)
		{
		gen_dist = op_dist_load ("normal",
msg_to_send[intrpt_code].frequency, msg_to_send[intrpt_code].variance);
		msg_to_send[intrpt_code].send_rate = op_dist_outcome
(gen_dist);
		}
} while (msg_to_send[intrpt_code].send_rate <= 0);	/* repeat until
the send rate is not equal to zero */

/* schedule another interrupt at this frequency */
```

```
op_intrpt_schedule_self (op_sim_time () + (double) 1.0/ ((double)
msg_to_send[intrpt_code].send_rate * 1/3600), intrpt_code);
```

OPNET CODE FOR AAAV_MSG_RCVR

Header Block
Deborah G. Peyton
May 1999

```
#include "tpal.h"

#define message_received (op_intrpt_type() == OPC_INTRPT_STRM)

/* define the structure of the parameters in the received message */
typedef struct
      {
      char msg_name[30];
      char destination[10];
      int size_of_data;
      int frequency;
      int variance;
      char distribution[15];
      double send_rate;
      } Msg_received;
```

State Variables Block

```
Objid \tpal_objid;

Ici * \ici_ptr;

int   \i;

/* PITCH_ANGLE_FDDI msg statistics collection for HEU-MPA to WSEU-FC */
/* arrival rate of message                                           */
Stathandle   \pa_rate_handle;

/* ROLL_ANGLE_FDDI msg statistics collection for HEU-MPA to WSEU-FC */
/* arrival rate of message                                          */
Stathandle   \ra_rate_handle;

/* time of last receipt of PITCH_ANGLE_FDDI Message */
double\pa_last_received_time;

/* time of last receipt of ROLL_ANGLE_FDDI Message */
double\ra_last_received_time;

/* PITCH_ANGLE_FDDI Msg arrival time */
double\pa_time_delta;

/* ROLL_ANGLE_FDDI Msg arrival time */
double\ra_time_delta;

Msg_received *     \msg_received;

int * \msg_count;
```

Temporary Variables Block

```
Packet*      pk_ptr;
```

INIT STATE

```
/*** Enter executive ***/

op_intrpt_schedule_self (op_sim_time(), 0);


/*** Exit executive ***/

/* initialize statistic handles */
pa_rate_handle = op_stat_reg ("PITCH_ANGLE Rate", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
ra_rate_handle = op_stat_reg ("ROLL_ANGLE Rate", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
```

PAUSE STATE

```
/*** Enter executive ***/

op_intrpt_schedule_self (op_sim_time(), 0);
```

REGSTR STATE

```
/*** Enter executive ***/

op_intrpt_schedule_self (op_sim_time(), 0);


/*** Exit executive ***/

// get TPAL object id
tpal_objid = op_topo_assoc (op_id_self(), OPC_TOPO_ASSOC_IN,
OPC_OBJMTYPE_MODULE, 0);

for (i = 1; i <= 4; i++)
      {
      ici_ptr = op_ici_create ("tpal_serv_reg");

      // set up TCP connection
      op_ici_attr_set (ici_ptr, "Protocol", "tcp");
      op_ici_attr_set (ici_ptr, "Port", i);
      op_ici_attr_set (ici_ptr, "Service Name", "FDDI Application-TCP");
      op_ici_attr_set (ici_ptr, "Popularity", 1.0);
      op_ici_install (ici_ptr);
      op_intrpt_force_remote (TPALC_CMD_SERV_REG, tpal_objid);

      op_ici_destroy (ici_ptr);

      ici_ptr = op_ici_create ("tpal_req");

      // listen on TCP connection
      op_ici_attr_set (ici_ptr, "flags", TPALC_OPT_PASSIVE);
      op_ici_attr_set (ici_ptr, "Remote Address", TpalC_Host_Unspec);
      op_ici_attr_set (ici_ptr, "Service", "FDDI Application - TCP");
      op_ici_attr_set (ici_ptr, "Remote Port", TpalC_Port_Unspec);
      op_ici_attr_set (ici_ptr, "Local Port", i);
      op_ici_attr_set (ici_ptr, "Protocol", "tcp");
      op_ici_install (ici_ptr);
      op_intrpt_force_remote (TPALC_CMD_OPEN, tpal_objid);
```

```
        op_ici_destroy (ici_ptr);
        }
```

WAIT STATE

```
/*** Enter executive ***/

//this state waits for a message to be received
```

RCV_MSG STATE

```
/*** Enter executive ***/

/* get the incoming packet and its attributes*/
pk_ptr = op_pk_get (op_intrpt_strm ());
op_pk_ici_get (pk_ptr);

/* extract the message data from the packet */
op_pk_nfd_get (pk_ptr, "Data", &msg_received);
op_pk_nfd_get (pk_ptr, "PUIT", &msg_count);

/* not all messages have names since all nodes are not attaching message
information - only the HEU*/
/* the following process will abort the program if a message name has not
been identified */
/* therefore, if the msg_name = NULL, skip the following process */
if ((msg_received->msg_name != OPC_NIL) && (msg_count != 0))
        {
        if ((strcmp(msg_received->msg_name, "PITCH_ANGLE_FDDI") == 0))
                {
                pa_time_delta = op_sim_time () - pa_last_received_time;
                pa_last_received_time = op_sim_time ();
                op_stat_write (pa_rate_handle, 1.0/pa_time_delta);
printf ("PITCH msg received at time %g with rate  %g\n", op_sim_time(),
1.0/pa_time_delta);
                }

        else if ((strcmp(msg_received->msg_name, "ROLL_ANGLE_FDDI") == 0))
                {
                ra_time_delta = op_sim_time () - ra_last_received_time;
                ra_last_received_time = op_sim_time ();
                op_stat_write (ra_rate_handle, 1.0/ra_time_delta);
printf ("ROLL msg received at time %g with rate  %g\n", op_sim_time(),
1.0/ra_time_delta);
                }
        }


/*** Exit executive ***/

/* destroy the packet */
op_pk_destroy (pk_ptr);
```

LIST OF REFERENCES

1. "USMC AAAV Program Brief." [http://www.aaav.hqi.usmc.mil/Program_Brief.ppt].

2. Telephone conversation between Major H. Oldland, USMC (DRPM AAA), Mr. C. Lamond, (DRPM AAA), Mr. Dinos Tsagos (DRPM AAA), Dr. B. Johnson (University of Virginia), Dr. R. Bernstein (Naval Postgraduate School) and the author, 26 February 1999.

3. "Direct Reporting Program Manager Advance Amphibious Assault Web Site." [http://www.aaav.hqi.usmc.mil/mainhome.html].

4. General Dynamics Land Systems, "Advanced Amphibious Assault Vehicle." [http://www.gdls.com/programs/aaav.html].

5. U.S. Marine Corps Specification RDD-P1010001A, *Advanced Amphibious Assault Vehicle Personnel Variant Requirements Definition Document for the AAAV-P Vetronic System Physical Architecture*, pp. 1-3, General Dynamics Amphibious Systems, November 1997.

6. U.S. Marine Corps Specification ICD-P1011402D, *Advanced Amphibious Assault Vehicle Personnel Variant Interface Control Document for FDDI Bus Build 3.0*, pp. 13-22, General Dynamics Amphibious Systems, 29 March 1999.

7. Peterson, L.L. and Davie, B.S., *Computer Networks: A System Approach*, pp. 292-3, Morgan Kaufman Publishers, Inc., San Francisco, CA, 1996.

8. Jain, R., *FDDI Handbook: High Speed Networking Using Fiber and Other Media*, pp. 379-424, Addison-Wesley Publishing Company, Reading, MA, 1994.

9. U.S. Marine Corps Specification ICD-P1011403D, *Advanced Amphibious Assault Vehicle Personnel Variant Interface Control Document for U.S.M.C. AAAV Utility Bus Build 2.2*, pp. 2-11, General Dynamics Amphibious Systems, 28 August 1998.

10. U.S. Marine Corps Specification RDD-P1011406, *Advanced Amphibious Assault Vehicle Personnel Variant Requirements Definition Document for Utility Bus Management Build 2.2*, p. 1, General Dynamics Amphibious Systems, 17 July 1998.

11. Email from Mr. D. Shakley (MITRE), Subject: AAAV Simulation Input, 9 April, 1999.

12. U.S. Marine Corps Specification ICD-P1011404B, *Advanced Amphibious Assault Vehicle Personnel Variant Interface Control Document for the AAAV-P Powertrain Data Bus Messages (CANbus) Build 2.2*, General Dynamics Amphibious Systems, 1 September 1998.

13. MTU Motoren- und Turbinen-Union Friedrichshafen GmbH, *Advanced Amphibious Assault Vehicle Interface Specification Control and Diagnostic System (CDS/CR)*, 27 July 1998.

14. SAE International Standard 1939/11, *Physical Layer--250K bits/s, Shielded Twisted Pair*, December 1994.

15. OPNET, *OPNET Modeling Manual*, Volumes 1 and 2, MIL 3, Inc. 3400 International Drive NW, Washington D.C., 20008, 1998.

16. U.S. Marine Corps Specification ICD-P1011402D, *Advanced Amphibious Assault Vehicle Personnel Variant Interface Control Document for FDDI Bus Build 3.0*, Appendix A, General Dynamics Amphibious Systems, 29 March 1999.

17. OPNET, *OPNET Models/Protocols*, MIL 3, Inc. 3400 International Drive NW, Washington D.C., 20008, 1998.

18. Email from Mr. D. Shakley (MITRE), Subject: FDDI MSG, 20 April, 1999.

19. Email from Mr. D. Shakley (MITRE), Subject: AAAV Simulation Input, 15 April, 1999.

20. Telephone conversation between Major H. Oldland, USMC (DRPM AAA), Mr. C. Lamond, (DRPM AAA), Mr. Dinos Tsagos (DRPM AAA), Dr. B. Johnson (University of Virginia), Dr. R. Bernstein (Naval Postgraduate School) and the author, 30 April 1999.

21. Email to Major H. Oldland (DRPM AAA), Subject: Scenario Update, 28 April 1999.

22. Email to Major H. Oldland (DRPM AAA), Subject: Response to Action Items from Telecon 3/26/99, 26 March 1999.

23. Stallings, W., *Data and Computer Communications*, pp. 420-1, Prentice Hall, Inc., Upper Saddle River, NJ, 1997.

24. FDDI Consortium of the Interoperability Lab at the University of New Hampshire, "FDDI Tutorial."

25. Standfield, R.D., *OPNET Implementation of Spread Spectrum Network for Voice and Data Distribution*, pp. 57-65, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1997.

26. Batson, M.S., *Loss Performance in an ATM Cell Capture Environment*, pp. 13-33, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1998.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code EC...1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

4. Professor Raymond F. Bernstein, Code EC/Be 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

5. Director, Marine Corps Research Center..1
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

6. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   300 Russell Road
   Quantico, VA 22134-5130

7. Marine Corps Representative...1
   Naval Postgraduate School
   Code 037, Bldg. 234, HA-220
   699 Dyer Road
   Monterey, CA 92943

8. DRPM AAA..2
AAAV Technology Center
991 Annapolis Way
Woodbridge, VA 22191

9. Robert E. Parker, Jr., LCDR/USN 1
2 University Circle, SMC #1165
Naval Postgraduate School
Monterey, CA 93943-1165

10. Ms. Deborah G. Peyton...1
2418 Still Forest Road
Baltimore, MD 21208

11. Col. J. E. Vesely..1
MARCORSYSCOM
2033 Barnett Ave Suite 315
Quantico, VA 22134-5010